(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
8 February 2001 (08.02.2001)

PCT

(10) International Publication Number
WO 01/09690 A1

(51) International Patent Classification[7]: G05B 15/00

(21) International Application Number: PCT/US00/20828

(22) International Filing Date: 28 July 2000 (28.07.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/146,406 29 July 1999 (29.07.1999) US
60/149,276 17 August 1999 (17.08.1999) US

(71) Applicant: THE FOXBORO COMPANY [US/US]; 33 Commercial Street, Foxboro, MA 02035 (US).

(72) Inventors: BADAVAS, Paul, C.; 19 Rockpoint Road, Southboro, MA 01772 (US). HANSEN, Peter, D.; 59 Fiske Road, Wellesley, MA (US).

(74) Agents: POWSNER, David, J. et al.; Nutter, McClennen & Fish LLP, One International Place, Boston, MA 02110-2699 (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/09690 A1

(54) Title: METHODS AND APPARATUS FOR OBJECT-BASED PROCESS CONTROL

(57) Abstract: A control system has blocks or other components that facilitate validation of their own replacements, e.g., downloaded via e-commerce transactions. The system includes first and second process control components. The first component is coupled to a third process control component, with which it transfers information, e.g., as part of an active or ongoing control process. The second component can be, for example, an update or other potential replacement for the first component. The first and/or second components can effect substitution of the second component for the first. More particularly, they can effect coupling of the second component for information transfer with the third component and decoupling of the first component from such transfer with the third component. Preferably, such coupling and decoupling occur while the process control system remains active.

Methods and Apparatus for Object-Based Process Control

Background of the Invention

This application claims the benefit of priority of United States Patent Application Serial No. 60/146,406, filed July 29, 1999, entitled Process Control Objects, as well as of United States Patent Application Serial No. 60/149,276, filed August 17, 1999, entitled Methods and Apparatus for Process Control ("AutoArchitecture"), the teachings of all of the foregoing of which are incorporated herein by reference.

The invention pertains to control and, more particularly, to methods and apparatus for implementing process and other control systems at lower cost, with greater flexibility and robustness.

The terms "control" and "control systems" refer to the control of a device or system by monitoring one or more of its characteristics. This is used to insure that output, processing, quality and/or efficiency remain within desired ranges over the course of time. In many control systems, digital data processing or other automated apparatus monitor a device, process or system and automatically adjust its operational parameters or variables. In other control systems, such apparatus monitor the device, process or system and display alarms or other indicia of its characteristics, leaving responsibility for adjustment to the operator.

Control is used in a number of fields. Process control, for example, is typically employed in the manufacturing sector for process, repetitive and discrete manufactures, though, it also has wide application in utility and other service industries. Environmental control finds application in residential, commercial, institutional and industrial settings, where temperature and other environmental factors must be properly maintained. Control is also used in articles of manufacture, from toasters to aircraft, to monitor and control device operation.

Modern day control systems typically include a combination of field devices, controllers, workstations and other more powerful digital data processing apparatus, the functions of which may overlap or be combined. Field devices include temperature, flow and other sensors that measure characteristics of the subject device, process or system. They also include valves and other actuators that mechanically, electrically, magnetically, or otherwise effect the desired control.

Controllers generate settings for the actuator type field devices based on measurements from sensor type field devices. Controller operation is typically based on a "control algorithm" that maintains a controlled device at a desired level, or drives it to that level, by minimizing differences between the values measured values and, for example, a setpoint defined by the operator. Workstations, control stations and the like are typically used to configure and monitor the process as a whole. They are often also used to execute higher-levels of process control, e.g., coordinating groups of controllers and responding to alarm conditions occurring within them.

In an electric power plant, for example, a workstation coordinates controllers that actuate conveyors, valves, and the like, to move coal or other fuels to a combustion chamber. The workstation also configures and monitors the controllers that maintain the dampers that determine the level of combustion. The latter operate, for example, by comparing the temperature of the combustion chamber with a desired setpoint. If the chamber temperature is too low, the control algorithm may call for incrementally opening the dampers, thereby, increasing combustion activity and driving the temperature upwards. As the temperature approaches the desired setpoint, the algorithm incrementally levels the dampers to maintain the combustion level.

The design of control systems and specification of the control algorithms is typically performed using tools known as configurators. An exemplary such tool is provided with the I/A Series® (hereinafter, "IAS" or "I/A") systems, marketed by the assignee hereof. A graphical configurator, FoxCAE,® provided with those systems permits an engineer to model a process hierarchically and to define a control algorithm from that hierarchy. Once configuration is complete, the control algorithm is downloaded to the control devices. This may involve "compiling" the algorithm in order to convert it into code understood by the controllers and other control devices.

While prior art products such as the aforementioned ones by the Assignee hereof continue to meet success in the marketplace, there remains a need for advancement.

In view thereof, an object of this invention is to provide improved methods and apparatus for control. A related object is to provide such methods and apparatus as can be achieved with lower cost, greater flexibility and robustness.

Another object of the invention is to provide such methods and apparatus as facilitate the modeling of control processes by engineers and users alike.

A related object is to provide such methods and apparatus as facilitate the generation of higher-quality modeling software at lower cost and more widespread applicability.

A further object is to provide such methods and apparatus as can be used in process and other control systems.

Summary of the Invention

The foregoing are among the objects attained by invention which provides, in one aspect, an improved control device for a process or other control system. The device provides a virtual machine environment in which Java objects, or other such software constructs, are executed to implement control (e.g., to monitor and/or control a device, process or system). These objects, referred to herein as process control objects (PCOs), define blocks, which are the basic functional unit of the control. They also define the input, output and body parts from which blocks are formed, and the signals that are communicated between blocks. PCOs also define nested and composite groupings of blocks used to control loops and higher-level control functions.

By way of non-limiting example, a control system with devices according to the invention can have a workstation and controllers, each providing a Java virtual machine (JVM) environment. Executing on the devices are Java PCOs embodying their respective control functions and signaling. Thus, PCOs executing in each controller monitor and control sensors and actuators under that controller's purview. PCOs executing in the workstation monitor and control the controllers themselves (as well, perhaps, as monitoring the outputs of some of the field devices). Intelligent field devices in the control system may also execute PCOs, thereby, further distributing the control function and taking up tasks otherwise handled by the controllers and workstation.

Referential Communication Between Process Control Entities

Further aspects of the invention provide control devices as described above in which individual data, such as measurements, setpoints or other values, are communicated block-type PCOs by reference. To this end, only one object of each pair of objects between which a given datum is communicated stores the datum itself, e.g., by non-limiting example, in a data structure referred to as below as a "variable." The other block maintains only a reference, i.e., a pointer, address, symbolic or other reference, to the datum. In order to access the sole instance of the datum as between at least those two

blocks, the latter block interrogates -- or, if permitted, sets the value of -- the datum by using the pointer, address or other reference.

A given datum, according to further aspects of the invention, can be maintained in the block that is the logical and/or physical source of the measurement, setpoint or other value to which it pertains. For example, a PCO embodying an analog input (AIN) block maintains a data structure containing data measured by it. PCO blocks that use those measurements access that data structure and, thereby, the data, by reference.

The data structures embodying data can themselves be PCOs, e.g., by non-limiting example, instantiated from the "signal" class described in the detailed description, below. According to further aspects of the invention, in addition to storing specific measurements, setpoints or values (e.g., the "variables" described below), these data structures can maintain range, status, time stamps and/or other information pertaining to them.

Thus, for example, a "float" variable data structure in a PCO analog input block maintained in a thermocouple sensor device can have, in addition to a floating point value representing the temperature value measured by that device, range values representing permissible upper and lower values for the measured temperatures. It can also have status values representing signal quality (e.g., SEVA values of the type discussed elsewhere herein) and/or initialization state; limit status for signal limiting and connection status; timestamp values identifying when the value was last changed; among others. According to related aspects of the invention, the "limit status" can include a flag, e.g., a "publish" bit, indicating whether the subject datum has been communicated to other elements of the control system, e.g., by "publication."

An advantage of devices using PCOs that communicate data, i.e., establish "connections," in the manner described above is that they minimize the unnecessary duplication of data and the attendant cost of maintaining coherency. A further advantage is that they can more readily propagate variables and their attendant values, ranges, and so forth, throughout the system.

With respect to this latter point, a control system having devices as described above will typically implement a control scheme wherein a value generated (or measured) by single block is used by several downstream blocks. Thus, for example, a PCO block maintained by a thermocouple field device might be connected to a PCO control block executing in a controller that processes the thermocouple output to adjust a fuel intake

valve. It might also be connected to a PCO control block executing in an intelligent field device that adjusts a damper level.

The use of data structures as described above facilitate establishing common values, ranges, and so forth, e.g., for the thermocouple output, among all of the PCO blocks that use connections. That information is stored only in the source PCO, e.g., the thermocouple PCO. Hence, the risk of data loss or misinterpretation resulting from lack of coherency is minimized, as is the risk of incorrect scaling and the like among the blocks participating in the connections.

Control devices as described above can, according to further aspects of the invention, maintain unilateral and/or bilateral connections for the information that they exchange. Unilateral connections utilize a data structure as described above to store a single "forward going" data value (or set of values), along with its attendant range, status, limit status, time and other related information. These are typically used, for example, in connections to/from sensors and actuators.

The control devices can also utilize PCOs that establish bilateral connections maintaining two data values (or sets of values), along with attendant range, status, limit status, time and other related information. These can be used, for example, in connections in which a forward going data value is dependent on a backward going one.

For example, a PCO functioning as a proportional integral derivative (PID) control block and executing in an intelligent actuator might provide the setpoint to a PCO executing as an analog output (AOUT) control block in that same actuator. A single data structure maintaining the AOUT setpoint can be stored in the PID, with the AOUT accessing it by reference. The PID requires feedback, such as the current valve position, in order initialize its output to avoid bumping that value at startup. The AOUT can provide that feedback, according to related aspects of the invention, by accessing the data structure by reference and storing a backward going data value (or feedback value) there. Similar bilateral communications are required between cascaded controllers where the output of one is the setpoint of another.

Control devices utilizing PCO that establish bilateral connections have additional advantages, including, eliminating the need to establish and ensure that forward-going and back-going values for each connection necessarily run between the same two blocks. In addition, they ensure that consistent range, status, limit status, time and other

information concerning the forward-going and back-going values are shared by both PCO that are parties to the connection.

Process Control Entities with Mandatory and Optional Parts

Further aspects of the invention provide control devices as described above that include PCOs with mandatory parts for which memory space is allocated at the time of object creation (or instantiation) and with optional parts for which memory space is allocated only as needed. The optional parts can be added subsequent to creation, typically, for example, during configuration.

By way of example, an intelligent field device embodying a PCO according to the invention representing an analog input block (AIN) can have a mandatory input part for receiving, linearizing, filtering and scaling measurements. It can also have a mandatory output part for processing or switching and for making the result available to PCOs, e.g., embodying control algorithms. These mandatory parts are instantiated when the AIN PCO is first instantiated. Optional parts for the AIN PCO permit, *inter alia*, establishing alarm limits, defining a characterizer that linearizes an input measurement, defining filtering for an input measurement, and defining limits for output values, and to utilize potential emergency interlock output values. These optional parts can be instantiated, if at all, e.g., when the already-instantiated PCO is being configured.

The invention provides, in other aspects, control devices as described above in which the parts, whether optional or mandatory, are associated with block-type PCOs and are instantiated locally in relation to the respective blocks that contain them. Put another way, the parts are instantiated in the processes responsible for executing the PCO in which they are contained.

Further aspects of the invention provide control devices as described above in which mandatory parts of a PCO are instantiated in a declaration or a constructor method (e.g., a default constructor) of a class from which the PCO is instantiated. While optional parts can be instantiated by a constructor (e.g., other than the default constructor), they can also be instantiated by configurator following creation of the PCO.

Control device incorporating dynamically configurable PCOs, i.e., with mandatory and optional parts, as described herein are advantageous, for example, in that their constituent blocks provide all necessary input, output and/or processing behaviors, without dedicating memory or other resources to unused ones. Thus, the AIN PCO

described in the example above can be selectively configured to allocate memory and processor resources to alarms and filters, yet, not to consume resources with optional features that will not be used, e.g., interlocks and characterizers. Moreover, such PCOs instill in their respective control devices (e.g., their respective control stations, work stations, controllers, or field devices) optional behaviors that execute as if "compiled in," yet, not requiring recompilation on configuration.

In addition to individual block-type PCOs that are dynamically configurable, further aspects of the invention provide control devices and systems with dynamically configurable *composite* PCOs. These are PCOs with mandatory constituent blocks for which memory space is allocated at the time of object creation (or instantiation) and with optional blocks for which memory space is allocated only as needed. The optional blocks are added at the time each composite PCO is created, e.g., during configuration, or later.

By way of example, a control system executing a PCO composite-type object representing a process control loop, can have a mandatory analog input block (AIN), a mandatory analog output (AOUT) block and a mandatory proportional-integral-derivative (PID) control block. Optional blocks for the composite PCO can provide for feedforward control. These can include, for example, an optional second AIN block, e.g., for detecting disturbances in the controlled process, as well as for an optional, feed forward control block that generates additional control values to facilitate disturbance compensation in the first (feedback), PID block.

Process Control System with Blocks Having Common Input and/or Output Sections

Still further aspects of the invention provide control systems and control devices as described above with block-type and composite-type PCOs that use standardized classes (or other definitional software constructs) to define input and output parts that receive and transmit information on behalf of the PCOs. The classes provide common interfaces between interconnected blocks, as well as insuring common processing of information by them.

By way of example, PCO blocks executing on the workstations, controllers, intelligent field devices and other control devices in a control system according to the invention can have input and output parts instantiated from a common set of input and output classes, respectively. The PCOs can be of a variety of composite block types and individual block types, the latter including, for example, AnalogInput, AnalogOutput,

PID, and Feedback Tuner. The common set of classes from which their input and output parts are instantiated include, according to one aspect of the invention and by way of non-limiting example, cascaded floating point input, cascaded floating point output, unidirectional boolean input, unidirectional boolean output.

According to related aspects of the invention, the input and output parts of the block PCOs are created from possibly overlapping subsets of classes selected from a common set of standardized classes. Thus, for example, PCOs that embody PID control blocks include input and output parts defined with standardized cascaded floating point input and cascaded floating point output classes. PCOs that embody "user" control blocks also use the standardized cascaded floating point output classes, though their input parts are defined using a different standardized class, to wit, the unidirectional floating point input class.

Further aspects of the invention provide control systems and control devices as described above wherein the input and output parts, themselves, use standardized classes to define objects that reflect, by way of non-limiting example, state, status, mode, and option assignments shared by the respective blocks and their parts, as well as to define information that is communicated between the blocks. In addition to providing common methods for setting and getting values, these classes define methods for linking variables (and their constituent parameters) to establish connections between blocks.

By way of example, the standardized cascaded floating point input part contained in the above-described PID control blocks can have constituent objects defined from a standardized "lrs" setpoint mode class that characterizes the setpoint source for an input part, e.g., whether it is set by an external block, by a supervisory task, or by the operator. By way of further example, the cascaded floating point output part of such a PID block can include a constituent objects defined from a standardized "mas" mode class that characterizes source of the outgoing signal generated by that output part, e.g., whether it is set by the block, a supervisory task or the operator.

A control device with a PCO block having input and output parts utilizing objects instantiated independent classes, such as the lrs and mas classes described above, can provide for the independent characterization of setpoints received and generated by the PCO. Thus, for example, the block can be set by an operator at runtime to provide for any of local, remote or supervisory setpoint input and, at the same time, for any of manual, automatic and supervisory setpoint output. Moreover, the block can retain a

setpoint value previously defined, e.g., before transition into manual or supervisory mode, to facilitate transition back to automatic mode.

Further related aspects of the invention provide control systems and control devices where the input, output, body and other parts use other standardized classes. These can include, by way of non-limiting example, a signal quality status class, a maintenance status class, a limit indication and linking/setting permissions class, among others.

The invention provides, in still further aspects, control systems and devices as described above in which memory is allocated for mandatory constituent portions of a part at the time of creation and in which memory is allocated for optional portions only as necessary, e.g., during configuration. These portions can include PCOs defined in accord with the aforementioned signal classes, as well as in accord with other classes that make up the parts.

According to further related aspects of the invention, that parts of PCO control blocks utilized in control systems and devices can have the mandatory constituent that include, by way of example, the aforementioned lrs and mas setpoint mode classes of the cascaded floating point input and output parts, respectively. Further mandatory classes of these parts can include, by way of further non-limiting example, a floating point variable class for containing the setpoints received or set by the input and output parts, respectively. Optional classes for these parts can include a limit class (defined as a "parts" class in the discussion below) used to define high and low setpoint values for the input and output parts that include the limit class.

Still further aspects of the invention provide control systems and devices as described above in which the block-type PCOs include body parts, in addition to input and output parts. The body parts can, for example, embody attributes and methods that are unique to the particular block class, in addition to standardized parts such as for feedback tuning and deadtime compensation.

Multi-Input Analog Input Block for Process Control System

Further aspects of the invention provide a control device with an analog input block (AIN) coupled to accept readings from multiple sensors or other input devices and to generate an output based on one or more of those readings. The analog input block, which can be a PCO as described above, can take a reading from each of the multiple

sensors during each block processing cycle (BPC), e.g., each cycle during which the AIN is invoked (typically, along or in sequence with other blocks in a common control system).

Related aspects of the invention provide a control device configured as an AIN as described above that generates an output based on the multiple inputs every "block period," e.g., every period (typically multiple BPCs) in which the output of the AIN is updated. The output can be, by way of non-limiting example, minimum, maximum, median, weighted average (e.g., based on uncertainty values, such as SEVA, provided by the sensor as part of the variable data structure) or other selection or function of the multiple inputs.

Further related aspects of the invention provide a control device configured by a PCO as an AIN block as described above in which the input part of that block samples measurements received from its respective sensor every BPC, notwithstanding that the block period may run over several BPCs. Those measurements can be averaged with one another (e.g. by the input or body parts of the PCO), prior to being statistically compared or combined with the outputs of other input parts in the AIN.

Still further aspects of the invention provide a AIN PCO as described above having multiple input parts, each receiving, filtering, characterizing and/or otherwise processing a respective one of the measurements received by the AIN block. A control device in which such an AIN is embodied can dedicate memory space upon instantiation of the block or its subsequent configuration to only such input parts as are required by the particular implementation. Moreover, such a control device can apply independent filtering, characterizing or other functions to each of the inputs.

Further aspects of the invention provide control systems embodying one or more control devices as described above and/or utilizing PCOs as described above. Other aspects provide control devices and systems having individual ones of the features described above, alone or in combination with other ones of the features.

Yet other aspects of the invention provide methods of operating control devices and control systems paralleling the foregoing.

These and other aspects of the invention are evident in the attached drawings, and in the description and claims that follow.

Brief Description of the Drawings

A more complete understanding of the invention may be attained by reference to the drawings, in which:

Figure 1 depicts a process control system 10 of the type with which the invention is practiced;

Figure 2 depicts the object model notation utilized in this disclosure;

Figure 3 depicts a control and information collection class and object architecture in a control system according to the invention;

Figure 4 depicts control and information collection signal, part, and block classes in a control system according to the invention;

Figure 5 depicts input/output classes for intelligent and standard devices in a control system according to the invention;

Figure 6 depicts control and information collection nested classes in a control system according to the invention;

Figure 7 depicts an architecture for a typical block class in a control system according to the invention;

Figure 8 shows examples of unidirectional and cascade input and output part classes in a control system according to the invention;

Figure 9 depicts a loop composite class in a control system according to the invention;

Figure 10 depicts a PLoop composite class in a control system according to the invention;

Figure 11 depicts a cascade composite class in a control system according to the invention;

Figure 12 depicts a TempProcess Composite class in a control system according to the invention;

Figure 13 depicts Loops2 control and process objects in a control system according to the invention;

Figure 14 depicts Loops2 control and process objects and details in a control system according to the invention;

Figure 15 depicts TempCasc control and process objects in a control system according to the invention;

Figure 16 depicts TempCasc control and process objects and details in a control system according to the invention;

Figure 17 depicts cascade peer-to-peer communication in a control system according to the invention; and

Figure 18 depicts variable and parameter transfer and rules in a control system according to the invention.

Detailed Description of the Illustrated Embodiment

Figure 1 depicts a process control system 10 of the type with which the invention is practiced. The system includes networked control devices that monitor and control a hypothetical mixing process 20 that utilizes mixing chamber 22, fluid inlets 24, 26, fluid outlet 28, paddle 30, cooler 32, and cooler inlet 34. Though illustrated and described below for use in connection with process control, those skilled in the art will appreciate that apparatus and methods according to the invention can be used in connection with any industrial, manufacturing, service, environmental or other process, device or system amenable to monitoring or control (hereinafter, collectively, "control").

The networked control devices include actuators, such as the valves depicted as controlling inlets and outlets 24 - 28 and 34. A further actuator is shown controlling paddle 30. These and other actuators utilized by the control system are constructed and operated in the conventional manner, as modified in accord with the teachings hereof. The actuators operate under control of respective field device controllers, labeled CTL, that are also constructed and operated in the conventional manner to provide initialization, signal conditioning and communications functions.

Rather than using separate controllers CTL, the actuators can be of the intelligent variety and can include integral microprocessors or other digital data processing apparatus for control, initialization, signal conditioning, communications and other control-related functions. For sake of convenience, the label CTL is used regardless of whether the control-related functionality is integral to the actuators (e.g., as in the case of intelligent actuators) or otherwise.

Illustrated sensor 29 monitors a temperature, level or other characteristic of fluid in chamber 22. The sensor 29, as well as other sensing apparatus utilized by the system, are constructed and operated in the conventional manner known in the art, as modified in accord with the teachings hereof. They can be coupled to the control network via a

transmitter or other interface device INT that, too, is constructed and operated in the conventional manner, as modified by the teachings hereof. The interface devices facilitate initialization, signal conditioning and communications between the sensors and the control system. As above, one or more sensors can be of the intelligent variety, incorporating integral microprocessors or other digital data processing capabilities for initialization, signal conditioning, communications and other control-related functions. Here, too, the label INT is used in reference to the control-related functionality, regardless of whether embodied in an intelligent transmitter or otherwise.

The networked control devices include one or more controllers 36 that monitor and control respective aspects of the hypothetical mixing process in the conventional manner, as modified in accord with the teachings hereof. The controllers can comprise mainframe computers, workstations 40, personal computers 42, special-purpose hardware or other digital data processing apparatus capable of performing conventional monitoring and control functions. Preferred controllers are constructed and operated in the manner of the CP control processors commercially available from the assignee hereof, as modified in accord with the teachings herein.

The control system 10 includes elements that serve as user interfaces and that provide configuration and/or control functions, all in the conventional manner as modified in accord with the teachings hereof. Illustrated for these purposes are, for example, workstation 40 and personal (laptop) computer 42. These devices can provide configuration and control functions directly, as in the case of workstation 40, or in cooperation with server devices (not shown). Apparatus 36 - 42 can couple with one another directly, e.g., via bus or network connection (as illustrated below), or indirectly, e.g., via satellite, wireless connection or modem connection.

The devices 36 - 42, CTL and INT, collectively, referred to as "control" devices, are coupled for communications via a medium that permits at least selected ones of the devices to communicate with one another. To this end, in the illustrated embodiment those devices are coupled via one or more networks 48 that are, preferably, IP-based such as, by way non-limiting example, Ethernets. The network(s) can include, as indicated by the multiple segments shown in the drawing, multiple segments such as various wide and local area networks. They may also include high and/or low bandwidth components, such as phone lines, and low and/or high latency components, such as satellites networks.

In the preferred, illustrated embodiment, each of the control devices 36 - 42, CTL and INT, provides virtual machine environment for executing Java byte code (or other such intermediate code) in the form of Java applications, Java applets, Java servlets, or like constructs in other software languages that, for example, configures the respective device to provide monitoring and/or control (collectively, "control"), including, by way of non-limiting example, servicing sensors to provide inputs (analog or otherwise), servicing actuators to provide outputs, executing blocks (composite, individual or otherwise) that include a control algorithm and/or otherwise to participate in the control system.

By way of non-limiting example, the illustrated intelligent field devices can include low power processors, along with a random access memory, read-only memory, FlashRAM, and appropriate sensor/actuator interfaces. The processors of those devices can execute a real-time operating system, as well as a Java virtual machine (JVM). Process control blocks or entities in Java byte code execute in the JVMs to configure the respective field devices to perform process control functions, e.g., for analog input (AIN), analog output, proportional integral derivative (PID) control and so forth.

By way of further non-limiting example, the controllers, workstations, control stations and other digital data processor-based control devices can include larger, more powerful central processing units, along with on-board memory (e.g., RAM, ROM, FlashRAM), mass storage, sensor/actuator interfaces, as necessary, conventional operating systems and JVMs, and so forth. The JVMs on these more powerful devices permit them to perform a still wider range of control functions, e.g., to monitor control larger scale plant operations.

A further appreciation of the hardware and software environment provided by system 10 may be attained by reference to commonly-assigned, co-pending United States Patent Application Serial No. 09/591,604, and to counterpart PCT Application Serial No. PCT/US 00/15860, both filed June 9, 2000, entitled METHODS AND APPARATUS FOR CONTROL USING CONTROL DEVICES THAT PROVIDE A VIRTUAL MACHINE ENVIRONMENT AND THAT COMMUNICATE VIA AN IP NETWORK, a copy of the common specification of which is attached hereto and which is reprinted as an Appendix hereof. The teachings of these applications are incorporated herein by reference.

Described below are aspects of the operation of the aforementioned control devices 36 - 42, CTL and INT, and, more generally, of the control system 10 deriving

from execution on the devices of Java applications, applets, servlets (or other such software constructs) to implement control. These objects, referred to herein as process control objects (PCOs), define blocks which are the basic functional unit of the control. They also define the input, output and body parts from which blocks are formed, and the signals that are communicated between blocks. PCOs also define nested and composite groupings of blocks used to control loops and higher-level control functions.

The Advantages of PCOs

As evident in the discussion above and in the pages that follow Process Control Objects facilitate the development, implementation and maintenance of control systems, additionally, maximizing their flexibility of configuration and robustness. PCOs also facilitate the introduction of new control objects and enhancements without affecting the running system, in addition to enabling on-line modification and upgrade of objects. Control systems utilizing PCOs are, moreover, portable: control classes, written once, can run everywhere. The systems are also scaleable.

By way of both review and introduction, control systems implemented with PCOs implement optional parts and functionality separately from mandatory parts, thus, minimizing consumption of undue resources. Their use of separate input and output parts speeds implementation, while permitting the input and output sides of each block to be independently configured and operated.

The PCO control architecture provides standardized input and output part classes used in all blocks. Input and output part classes provide initialization, anti-windup, cascade handling, local/remote/supervisory switching, manual/auto/supervisory switching, standardized variable naming, input and output multiplicative and additive feedforward, output selection (limiting with an override controller). In addition optional parts for the input and output classes furnish alarming, limiting, filtering, feedforward, characterization, and interlock. Architecturally the input and output part classes provide standard interfaces between interconnected blocks.

PCOs provide a framework where interconnected block and nested classes are aggregated to create higher level nested classes, which are then used to create object instances. These higher nested instance include, for example, all control for a process unit, all information collection for a process unit, and so forth. Information is stored at its source, permitting other objects to reference it, e.g., via implicit pointers, addresses,

symbolic references, or the like, in order to obtain specific datum (such as measurements) and information pertaining to it. For objects residing in remote stations the source passes the referenced information to proxy objects residing in those stations.

In a control system implemented using PCOs, scaling is done at the source and transmitted to the sink(s) when it changes. For example, in a cascaded controller, the local, remote, and supervisory setpoints obtain scaling from the measurement input. A cascade output obtains scaling from its back-calculated input. In general, cascade scaling information flows upstream. An analog output object has no measurement, therefore the scaling of its remote setpoint and back-calculated output is either entered locally or obtained from the downstream field device.

Other aspects of a control implementation using PCOs is that it permits the creation of object classes for required inter-block signals to communicate values with signal (SEVA) quality status, cascade handling (bumpless initialization and anti-windup) logic, and permissions for connection, setting, and configuration. Moreover, it makes automatically the backward interconnection for cascaded blocks.

Object Model Notation

Figure 2 shows the notation for object and class diagrams used in this patent application. Those skilled in the art will appreciate that this is a subset of the standard Rumbaugh notation.

Architecture of Process Control Classes

This section describes the overall architecture for Process Control Objects and introduces the signal, part, block, nested, and composite classes and objects.

Signal Object. A signal object is an object that is implemented as a component of a block or a part object, and its name includes the name of its block object. It is located locally in relation to the block object that contains it. A signal object is an instance of a signal class. Signal objects contain the information communicated between blocks or inserted into blocks as constants. These objects also provide methods to be used by users and tasks for setting and getting values and for linking and unlinking parameters and variables.

Signal Classes. Signal Classes are classes that provide signal objects as defined above.

Part Object. A part object is an object that is implemented as a component of a block or another part object, and its name includes the name of its block object. It is located locally in relation to the block object that contains it. A part object is an instance of a part class. Input and output part objects contain or reference float, boolean, or integer signal objects, which enable uni-directional and cascade communication between blocks. Some part classes provide objects for optional functionality such as alarm, limit, and filter, which can be components of the input and output part objects or blocks.

Part Classes. Part Classes are classes that provide part objects as defined above.

Block Object. A block object is an instance of a block class and provides a base level of functionality. It is the lowest level object, which can be implemented as an independent object with a network-wide unique name. A block object is made up of input part objects, a body, and output part objects. The input and output parts are independent of the body and enable uni-directional and cascade communication between blocks. The body contains the attributes and methods that are unique to the particular block class. In addition the body may contain optional parts such as feedback tuner, feedforward tuner, and deadtime. AnalogInput, AdvancedPID, and AnalogOutput are examples of block objects. Typically, block objects are used to create a hierarchy of nested objects.

Block Classes. Block Classes are classes that provide block objects as defined above.

Nested Object. A nested object is an instance of a nested class and is a component of possibly interconnected block and nested objects. Replication of a control solution is achieved with multiple instances of a nested class. Each object instance can be implemented as an independent object with a network-wide unique tag name. An example of a nested class is a loop class, name it Loop, which contains the AnalogInput, AdvancedPID, and AnalogOutput block classes with appropriate connections achieved by linking the input and output parts of the blocks. Class Loop can then be used to create many instances of loop composite objects. A second level nested class can be created by nesting together the AnalogInput and AdvancedPID block classes and the Loop class with appropriate connections to create a cascade class, name it Cascade. Class Cascade can then be used to create many instances of cascade composite objects.

Nested Classes. Nested Classes are classes that provide nested objects as defined above.

Composite Object or Composite. A composite object is an instance of a nested class implemented as an independent object with a network-wide unique name. For example instances of class Loop or Cascade with network-wide unique names are composite objects.

Composite Class. A composite class is a nested class whose object instances are implemented as independent objects with a network-wide unique name, that is, composite objects. When class Loop is used to create instances with network-wide unique names it is a composite class. However, the same class Loop when used as a component of the Cascade class is a component class, while class Cascade is a composite class.

Figure 3 shows the architecture for the Process Control Objects (PCO). Square boxes indicate classes, while rounded ones indicate object instances (objects). The Signal Classes, shown with vertical lines, provide signal classes to the part and block classes, for example the float parameter and float variable classes.

The Part Classes, shown shaded, provide mandatory and optional part classes to the block classes, for example the uni-directional and cascade float input and output classes. The optional part classes also provide optional part classes to the mandatory part classes.

The Block Classes are typically used to create component and composite classes, which are then used to create many instances of composite objects (composites).

A tag name is an arbitrary string.

Predefined (Implementation-Standard) Control Classes

Figure 4 shows the signal, part, and block classes for control and information collection in more detail. In particular it shows the inheritance hierarchy and aggregation of the part and block classes for control and information collection.

Class names start with a capital letter, while names for objects, attributes, and methods start with small letters with additional words concatenated to the first word starting with capital letters. This is the recommended practice by many authors and developers and is followed in this document.

The clear rectangular boxes indicate block classes, for example, AnalogInput class. The shaded rectangular boxes indicate part classes, for example, Alarm and FloatVariable classes. The boxes with the diagonal lines indicate signal classes, for example, boolean and float parameters and variables.

Figure 5 shows the InputOutput classes for intelligent and standard devices, which are indicated by boxes filled with dots. The boxes with dotted lines indicate classes from Figure 4. The InputOutput classes are parts of the UniDirFloatInput and CascFloatOutput classes for analog inputs and analog outputs, respectively.

Figure 6 illustrates the aggregation of control block classes to create many levels of nested classes. The inheritance hierarchy and aggregation classes for control part and block objects shown in Figure 4 furnishes the control block classes used to create nested classes.

As shown in the figure, the objects of class ControlNestedLevel1 contain block objects. In particular, they contain combinations of two or more (indicated by 2+) block objects. The objects of class ControlNestedLevel2, contain combinations of one or more (indicated by 1+) ControlNestedLevel1 objects, and optionally zero or more block objects. A level 2 control nested object must contain at least one level 1 control nested object.

Following this procedure, nesting can be generalized to N levels. Each higher level control nested object contains at least one control nested object from the level below it, and optionally, zero or more objects from any other levels as well as zero or more block objects.

This is how the composite classes of Figure 3 are created and then used to create many instances of composite objects as required.

Figure 7 shows the architecture for a typical block class. The Input Parts contain the parts and references for uni-directional and cascade variables. The input parts can also contain optional parts such as filter and alarm. The Output Parts contain parts and references for unidirectional and cascade variables. The output parts can also contain optional parts such as limit and alarm. The Body contains the attributes and methods that are unique to the particular block class. It can also contain optional parts such as feedback tuner and deadtime.

Figure 8 provides examples of uni-directional and cascade input and output parts. More particularly, Figure 8(a) shows the uni-directional part UniDirInput. The variable *in* comes from a source and there is no backward communication as indicated by the single incoming arrow. The variable *pv* can be a source for many sinks and there is no backward communication as indicated by the single outgoing arrow.

Figure 8(b) shows the cascade part CascadeInput. The cascade part enables uni-directional and bi-directional communications. The variables *in, fdFwd, pv, sp,* and *deviation* are uni-directional as indicated by the single arrow. Variables *remSP and supvSP* are bi-directional as indicated by the double arrow.

Similarly, Figure 8(c) and (d) shows the uni-directional part UniDirOutput and the cascade part CascadeInput, respectively.

The Input and Output parts are instances of the part classes that are shown in the Control and Information Class structure of Figure 4 and they are separate and independent from the Body of the typical block class. This enables inter-operability between components by connecting the output variable of an output part to the input variable of an input part.

When either the output part of the source block or the input part of the sink block is uni-directional, the communication between them is uni-directional (one- way) from the source to the sink. For example, the value and status of the output part's variable is passed by reference to the input part's variable.

When both the output part of the source block and the input part of the sink block are cascade, the communication between them is also cascade. That is from the source to the sink for both the forward value and status, and from the sink to the source for both the back value and status. In this case the sink component is connected to the source component and the source component is connected to the sink component. A single linkage of the sink to the source connects both the forward and back signals.

Control applications use both uni-directional and cascade connections. Consider, for example, heating cold water in a heat exchanger by condensing steam. The cold water flows through the tubes of the heat exchanger and steam flows through the shell of the heat exchanger. The objective is to heat the water to a desired temperature using cascade control, whereby the temperature controller sets the setpoint of the steam flow controller. The temperature controller uses the hot water temperature measurement, while the steam flow controller uses the steam flow measurement. The steam flow controller delivers the required steam by throttling the steam valve.

This is an example of cascade control, that is, the output of the temperature controller is cascaded to the setpoint of the steam flow controller. Consider the case where the operator opens the cascade by placing the steam flow controller on local control, and sets the steam flow setpoint herself, thereby disabling the temperature

controller from doing so. When the operator closes the cascade and transfers control to the temperature controller there is going to be a "bump" in the steam flow unless the temperature controller output was initialized to the value of the steam flow local setpoint at the time of transfer.

This is precisely accomplished with the cascade connection. In normal cascade control the output of the temperature controller is connected to the setpoint of the steam flow controller. The forward value and status of the steam flow controller setpoint are the value and status of the temperature controller output to which it is connected. Also the back value and status of the temperature controller output are the back value and status of the steam flow controller remote setpoint to which it is connected.

When the cascade has been open for a period of time and the operator transfers control to the temperature controller by switching the steam flow controller from local to remote, the temperature controller knows this from its output's back status from the steam flow controller setpoint. At this time the temperature controller initializes and computes the value of its forward output to be the output back value, which the steam flow controller sets equal to its local setpoint value at the time of transfer. From then on normal cascade control commences again and the temperature controller manipulates the steam flow setpoint to drive the actual water temperature to the desired value, as determined by the value of the temperature controller setpoint.

Signal Classes

The signal classes are contained in the Signals package, described below. A package contains a number of Java files and each Java file contains one and only one class. Objects from Signals classes contain the information communicated between linked blocks or inserted into blocks as constants. These classes also provide methods to be used by users and tasks for setting and getting values and for linking and unlinking parameters and variables. The Signals package also contains an interface for shared constants, and a mode class, which is used by cascade *input* parts for local/remote/supervisory switching and by cascade *output* parts for manual/auto/supervisory/interlock switching.

The signal classes include the following: BoolParameter, BoolVariable, FloatParameter, FloatRange, FloatVariable, IntParameter, IntRange, IntVariable, Mode, ObjList. Parameter and Variable are abstract classes passing common methods and attributes through inheritance to the individual XxxParameter and XxxVariable classes.

Part Classes

The part classes are contained in the Parts package. A Part must be contained in a block or other part contained in a block. Each block may have one or more *input* parts, a body, and one or more *output* parts. Input and output Part classes contain or reference FloatVariable, BoolVariable, or IntVariable signals, which enable uni-directional or cascade communication between blocks. Some part classes provide objects for optional functionality such as alarm, limit, and filter and they can be components of input and output part objects or blocks.

An unused optional part has a null reference. The configurator must instantiate an optional part before it can be used or its parameters set.

The parts classes include the following: Alarm, BoolFilter, Characterizer, CascBoolInput, CascBoolOutput, CascFloatInput, CascFloatOutput, CascIntInput, CascIntOutput, Deadtime, Failsafe, FeedForward, Filter, InfoBuffer, IntFilter, Interlock, Limit, LinearFlow, PIDParameters, Ramp, TimeCount, UniDirBoolInput, UniDirBoolOutput, UniDirFloatInput, UniDirFloatOutput, UniDirIntInput, UniDirIntOutput, Zone,

InputOutput Classes

The InputOutput classes are contained in the InputOutput package. The InputOutput classes are parts of the UniDirFloatInput and CasFloatOutput classes for analog inputs and analog outputs, respectively. They are optional parts with a default null reference. The configurator must instantiate them before they can be used or their parameters set.

The InputOutput Classes provide the interface to input and output signals using the Java Native Interface (JNI), which provides Java classes with native methods.

Block Classes

The block classes are contained in the Blocks package. The block classes provide block object instances, which can be implemented as independent objects with network-wide unique names. Typically, block classes are used to provide component classes for composite classes, which are then used to create many instances of composite objects.

Figure 7 shows the architecture for a typical block class. The Input Parts contain the parts for uni-directional and cascade variables. The input parts can also contain optional parts such as filter and alarm. The Output Parts contain the parts for uni-

directional and cascade variables. The output parts can also contain optional parts such as limit and alarm. The Body contains the attributes and methods that are unique to the particular block class. It can also contain optional parts such as feedback tuner and deadtime.

The Input and Output parts are instances of the part classes and are separate and independent from the Body of the typical block class. This enables inter-operability between components by connecting the output variable of an output part to the input variable of an input part.

The configurator must instantiate a block and its mandatory and selected optional parts before it can be used or its parameters set.

Block is a parent class (superclass) to all block classes including the UserBlock class and its attributes and methods are inherited by all block classes. The UserBlock is a parent class to the Arithmetic and UserDefined classes and its attributes and methods are inherited by them. The following two sections provide a detailed functional description of the Block and UserBlock classes.

Block Class

Overview

Block is an abstract class that contains data and methods common to all blocks. It implements the Constants and Runnable interfaces. Options, mode, and status are conveyed with packed Booleans. Bit-mask definitions are contained in the Constants interface, which is inherited by most classes.

A part class, which provides a common function, may be contained in several block classes or other part classes. An optional part may be instantiated at configure time. If it is not configured, it has a "null" value in the container object.

The *run* method of each block is called each *bpc*. Its output is updated each *period* at a specified *phase*. The *bpc* is a configured float number in seconds, an integer number of milliseconds, preferably an integer multiple of .1 seconds. The nominal *period* is an integer number of *bpc*s less than or equal to the configured float *blockPeriod*, but at least one *bpc*. The *execute* method determines when each block updates its output based on its configured *blockPeriod* and *phase*. The actual *period* is based on the time since the start of the last execution.

An output variable object is stored in an *output* part of its source block object. An input variable object of a block's *input* part may be linked (connected) to the output variable object by setting the input variable object equal to the output variable object. This is done at configure time with the output variable object's *linkTo* method, which checks for permissions.

Output variables are instances of the FloatVariable, BooleanVariable, or IntVariable classes. These contain a *forw* and may contain a *back* parameter object which are instances of the FloatParameter, Boolean Parameter, or IntParameter classes. Each of the parameter classes contains a value and two status bytes. The status bytes provide SEnsor VAlidation (SEVA) quality indications, cascade handling bits, downstream limit bits, and permissions. The parameter classes have *getStatus*, *getValue*, *setValue*, *linkTo*, and *unlink* methods to be used by users and tasks. The Variable classes contain alarmSum and maintenance status. The FloatVariable also contains *linkTo*, *ack*, and *getAlarm* methods, a range object, and the SEVA *'uncertainty'*. Both source and sink have access to the members of a linked Variable object.

There are two types of *input* objects. One is an instance of the UniDirFloatInput class, the other is an instance of the CascadeFloatInput class. The UniDirFloatInput class accepts one input variable object. It may contain charac and filter objects, used to precondition the input variable prior to its use in the block algorithm. It outputs the *pv* variable. The CascadeFloatInput class inherits from the UniDirFloatInput class and also receives local, remote, supervisory, and feedforward input variables in addition to the measurement input variable. It outputs *pv* and *deviation*. The CascadeFloatInput class does the local, remote, supervisory mode switching and cascade handling for bumpless transfer and initialization. It also may contain a setpoint *limit* object, a local setpoint ramp object, a deviation alarm object, and an additive or multiplicative *fdFwd* object.

There are also two types of output objects. One is an instance of the UniDirFloatOutput class, the other is an instance of the CascFloatOutput class. The UniDirFloatOutput class provides one output variable object and receives manual and supervisory input variable objects. It provides auto, manual, supervisory mode switching and may provide an output 'limit' object and two output alarm objects. The CascFloatOutput inherits from the CascFloatOutput class. It also provides bumpless mode transfer and initialization. It may contain an additive or multiplicative *fdFwd* object and a limit object.

Attributes

tag	object instance tag, a string.
description	object instance description, a string.
type	object instance type, a string
blockPeriod	is the time in seconds between output updates, float, default .5.
phase	object instance phase, a double integer, default is 0.

Methods

run	Arguments: none.
	Executes object instance when invoked.
	Returns: none.

UserBlock Class

Overview

An object from the UserBlock class has parts from the UniDirBoolInput, IntInput, and UniDirFloatInput classes, parts from the CascBoolOutput, CascIntOutput, and CascFloatOutput classes, and output mode parts from the Mode class.

Initialization, bumpless auto/manual/supervisory switching, and antiwindup protection are automatically provided when the user program calculates its *outputFloat[i].out.forw.value* by adding an incremental change to its *outputFloat[i].out.back.value*. Toggled booleans are treated similarly. The number of inputs, outputs, and modes are specified as the constructor arguments in the above order.

Methods from the FloatParameter, IntParameter and BoolParameter classes can be used *to getValue(), setValue()*, and *getStatus()*. Methods from the FloatVariable and BooleanVariable classes can be used to *getAlarm()* and *ack()*. Methods from the Mode class can be used to *getMode()* and *setMode()*. Definitions of status and modes are in the Constants interface.

Attributes

inputBool	is an optional array of BooleanInput parts.
outputBool	is an optional array of *CascBoolOutput* parts.
inputFloat	is an optional array of UniDirFloatInput parts.
outputFloat	is an optional array of *CascFloatOutput* parts.

numberFloatOut	number of float *outputFloats* specified as constructor argument.
mode	is an optional array of Mode parts.
seed = 100001	is an int seed number for generating the uniform (0,1) probability Distribution. Different seed numbers generate different, and statistically independent distributions.

Methods

run

Arguments:	none
Called	each time step to run the block.
Returns:	none

runMethod	contains the user written code.
Arguments:	none
Called	by run method
Returns:	none

ranUniform

Arguments:	none.
Computes	uniform random number when invoked.
Returns:	random number uniformly distributed between 0 and 1,

ranGauss

Arguments:	none.
Computes	Gausian random number when invoked.
Returns:	random number distributed normally, with mean = 0 and standard deviation (ó) = 1.0, also known as the Gaussian distribution.

Composite Classes

Figure 5 shows the implementation of composite classes in general. This section describes a library of the Nested superclass and composite classes.

Nested

Overview

Nested is an abstract class containing Strings for nested classes. It implements the Constants and Runnable interfaces. The *config* method prepares a *comp* list of its runnable components. The run method calls the run method of each composite and block on the *comp* list.

Attributes

tag_desc	is a *String* composite user tag
description	is a String composite brief description.
type	is a String identifying the composite class.
offOn	is a boolean that allows its components to run when true. Its default is false.

Methods

config

Arguments: none.

Returns: none.

run

Arguments: none.

Returns: none

Loop

Overview

Loop is a user defined nested class. It contains *config* and *run* methods, which are called by the *main* method.

Figure 9 shows the Loop composite class, which contains the AnalogInput, AdvancedPID, and AnalogOutput classes connected as shown. Each time the Loop class is used to create a Loop composite object it also creates an instance of *ain, pid,* and *aout*. Each Loop composite object is created with the default attributes of the *ain, pid* , and *aout*, however, the default attributes can be overridden.

Attributes

ain	is an instance of AnalogInput class.
pid	is an instance of AdvancedPID class.
aout	is an instance of AnalogOutput class.
comp	is a *ObjList* of *Runnable* objects.
sp	is the alias for the pid setpoint, a FloatParameter

pv is the alias for the pid process variable, a FloatParameter

out is the alias for the pid output, a FloatParameter

Attributes from superclass

tag user specified Loop tag, a String

description "Secondary", a String.

type "Loop", a String.

Methods

Methods from superclass

Config

Arguments: none.

Returns: none.

run

Arguments: none.

Returns: none

PLoop

Overview

PLoop is a user defined nested class. It contains *config* and *run* methods, which are called by the Configure class.

Figure 10 shows the Ploop (Primary Loop) composite class, which contains the AnalogInput and AdvancedPID classes connected as shown. Each time the PLoop composite class is used to create a PLoop composite object it also creates an instance of *ain* and *pid*. Each PLoop composite object is created with the default attributes of *ain* and *pid*, however, the default attributes can be overridden.

Attributes

ain is an instance of AnalogInput class.

pid is an instance of AdvancedPID class.

comp is a *ObjList* of *Runnable* objects.

sp is the alias for the pid setpoint, a FloatParameter

pv is the alias for the pid process variable, a FloatParameter

out is the alias for the pid output, a FloatParameter

Attributes from superclass

tag user specified Ploop tag, a String

description "Primary", a String.

type "Ploop", a String.

Methods from superclass

config specifies internal default values and connections.

Arguments: none.

Returns: none.

run schedules and runs contained components when run is called.

Arguments: none.

Returns: none

Cascade

Overview

Cascade is a user defined nested class. It contains config and run methods, which are called by the Configure class.

Figure 11shows the Cascade composite class, which contains the Ploop and Loop composite classes connected as shown.

Attributes

Pri is an instance of the Ploop class.

sec is an instance of the Loop class.

comp is a *ObjList* of *Runnable* objects.

attributes from superclass

tag user specified Cascade tag, a String

description "TempCascade", a String.

type "TempCascade", a String..

Methods from superclass

config specifies internal default values and connections.

Arguments: none.

Returns: none.

run schedules and runs contained components when run is called..

Arguments: none.

Returns: none

User Composite Classes

The user composite classes are examples of classes that would be created by a user in the same way as the composite classes discussed above. In addition users can create their own subclasses under the superclass UserBlock shown in Figure 4.

The remainder of this section describes the TempProcess composite class, which is used to create instances that simulate a process, which provides temperature and flow measurements.

TempProcess

Overview

Figure 12 shows the TempProcess user composite class, which contains two ProcessSim classes connected as shown. Each time the TempProcess composite class is used to create a TempProcess composite object it also creates an instance of *flow* and *temp*. Each TempProcess composite class is created with the default attributes of *flowOut* and *tempOut*, however, the default attributes can be overridden. The flow process is a one time step delay (.5 sec.) and the temperature process is a 10 sec. delay.

Attributes

flow is a ProcessSim block with 1 input.

temp is a ProcessSim block with 1 input.

comp is a *ObjList* of *Runnable* objects.

FlowOut is the output of the flow process, a FloatVariable.

TempOut is the output of the temp process, a FloatVariable.

Attributes from superclass

tag user specified TempProcess tag, a String

description is the String "Temperature Process".

type is the String "TempProcess".

Methods

config specifies internal default values and connections.

Arguments: none.

Returns: none.

run schedules and runs contained components when run is called.

Arguments: none.

Returns: none

Supv

Overview

Supv is a user defined class derived from the UserBlock class. It contains a run method. It has one floatOut and no inputs. It contains the controller parameters and ramps its output, used as the supervisory setpoint of the temperature controller.

Attributes

parTC is a PIDParameters part that contains parameters for the temperature controller.

ParFC is a PIDParameters part that contains parameters for the flow controller.

Attributes from superclass

tag user specified Supv tag, a String

description is the String "Supervisory Setpoint".

type is the String "Supv".

Methods from superclass

run

Arguments: none

Called each time step to run the block.

Returns: none

runMethod contains the user written code.

Arguments: none

Called by *run* method

Returns: none

Process Control Application Classes

The Block, Composite, and UserComposite classes can be used to create many object instances, which are simply referred to as composite objects or composites. An application instantiates and configures the composite objects using the *config* method and then executes them using the *run* method. The *config* method prepares a *comp* list of its runnable composites. The run method calls the run method of each composite on the *comp* list. The following two sections present the Loops2 and TempCasc applications.

Loops2

The Loops2 application illustrates composite objects for control and process simulation, which are made up of block objects.

Overview

Figure 13 shows the Loops2 control composite objects and the process composite object, while Fig. 8.4.1-2 shows the same composites in detail. Application Loops2 uses the *config* method to instantiate (configure) composite objects *s21, tl21, fl21,* and *pr21* and the *run* method to schedule them for execution.

Composite object *s21* (Supervisor 21) is an instance of the Supv block class described in Sec. 8.3.2. It acts as a supervisory component and its output *s21.out* sets the temperature setpoint *tl21.sp* of Temp Loop *tl21.*

Composite object *tl21* is an instance of the PLoop component class shown in Figure 10 and is a single loop made up of block instances *ain,* and *pid.* Temp Loop 21 compares the temperature measurement value *tl21.pv* to the temperature setpoint *tl21.sp* and computes *tl21.out,* which sets the required steam flow setpoint *fl21.sp*

Composite object *fl21* (Flow Loop 21) is an instance of the Loop composite class shown in Figure 9 and is a single loop made up of block instances *ain, pid,* and *aout.* Flow Loop 21 compares the steam flow measurement value *fl21.pv* to the required steam flow setpoint *fl21.sp* and computes the required steam valve position *fl21.out,* which sets process variable *pr21.pv*

Composite object *pr21* (Process 21) is an instance of the TempProcess composite class shown in Fig. 8.3.1-1 and simulates the temperature process. Process 21 uses process variable *pr21.pv* and computes the resulting steam flow *fl21.pv* and temperature *tl21.pv.*

Object *ic21* is an instance of the InfoCollect block class. Object *ic21* collects the date/time, value, and status for 10 (configurable) samples of *tl21.pv, tl21.sp, fl21.pv, fl21.sp,* and *fl21.out.* Each of these objects is double-buffered and when one buffer fills it switches to the other, which it overwrites.

Attributes

s21 is a Supv block.

tl21V1 is version 1 of a Ploop composite.

tl21 is an alias for a Ploop composite.

tl21V2 is version 2 of a Ploop composite

fl21 is a Loop composite.

pr21 is a TempProcess composite.

ic21 is an InfoCollect block.

comp is a *ObjList* of *Runnable* objects.

Methods

config overrides default values and makes external connections.

Arguments: none.

Returns: none.

run schedules Loop2 objects and calls their run methods.

Arguments: none.

Returns: none

TempCasc

The TempCasc application illustrates the *cascade* composite object for control, which is made up of component (composite) objects, while the process composite is made up of block objects.

Overview

Figure 13 shows the TempCascade composite object and process, while Fig. Figure 14 shows the same components in detail. Application TempCasc uses the *config* method to instantiate composite objects *s21, tc21,* and *pr21*, and the *run* method to schedule them for execution.

Composite object *s21* (Supervisor 21) is an instance of the Supv block class described in Sec. 8.3.2. It acts as a supervisory component and its output *s21.out* sets the temperature setpoint *tc21.sp* of Temp Loop *tc21.*

Composite object Temp Cascade 21, *tc21*, is an instance of the Cascade composite class shown in Fig. 8.2.5-3 and is a cascade loop made up of Ploop instance *pri*, and Loop instance *sec.*

Temp Cascade 21, *tc21*, uses measurement values *tc21.pv1* and *tc21.pv2* and temperature setpoint *tc21.sp* and computes the required steam valve position *out21.*

Composite object *pr21* (Process 21) is an instance of the TempProcess component class shown in Fig. 8.3.1-1 and simulates the temperature process. Process 21 uses process variable *pr21.pv* and computes the resulting steam flow*fl21.pv* and temperature *tl21.pv.*

Object *ic21* is an instance of the InfoCollect block class. Object *ic21* collects the date/time, value, and status for 10 (configurable) samples of *tc21.sp, tc21.pv1, tc21.pv2,* and *fl21.out*. Each of these objects is double-buffered and when one buffer fills it switches to the other, which it overwrites.

Attributes

s21 is a Supv block.

tc21 is a Cascade composite.

pr21 is a TempProcess composite.

comp is a *ObjList* of *Runnable* objects.

Methods

Config overrides default values and makes external connections.

Arguments: none.

Returns: none.

Run schedules TempCasc objects and calls their run methods.

Arguments: none.

Returns: none

Information Collection Objects

The purpose of information objects is to collect information for date/time, bool, int, and float parameters, alarmSum, and modes.

The purpose of InfoCollect objects is to collect a specified number of samples for the date/time, value, and status of a specified number of bool and float parameters, alarmsSum, and mode. The samples are collected in two buffers. When one buffer fills data storage is switched to the other.

An object from the InfoCollect class has *paramBool* parts from the BoolParameter class, paramInt parts from the IntParameter class, *paramFloat* parts from the FloatParameter class, actual *mode* parts from the Mode class, and *alarmSum* short integers.

Methods from the source FloatParameter and BoolParameter classes can be used to linkTo().

An object from the Event class may have a *paramBool* part from the BoolParameter class, paramInt parts from the IntParameter class, a *paramFloat* part from

the FloatParameter class, an actual *mode* part from the Mode class, and an *alarmSum* short integer.

Methods from the BoolParameter, IntParameter, and FloatVariable classes can be used to linkTo().

Configuration

All parts that a block or part may use are declared. Mandatory parts are instantiated in the declaration or constructor, while optional parts are not. If optional parts are used, they are instantiated by the configurator. An arbitrary number of a specific part type is specified as an argument of the container's constructor method. For example, see the Supv class derived from the UserBlock in Loops2. An example of an optional part class is Filter, which is used in the CascFloatInput class.

Block classes are the lowest level classes that can be used as component classes that are optionally interconnected to create composite classes. Of course the composite classes can contain not only block classes but also other composite (component) classes, and so on. The block classes and their mandatory and optional part, signal, and inputOutput classes contain default values for each attribute.

Configuration of Composite Classes

Java code is written and compiled to create a new composite class. An already existing composite class can be instantiated and the resulting composite object can be customized to suit the application without compiling.

The code for a new composite contains name declarations of component blocks and composites. Its constructor method preferably contains calls to the component's constructor methods. A block object that may have multiple part objects from the same part class requires that the number of such part objects be specified as an argument in the block's constructor. This number may be passed from a containing composite constructor's argument.

A *config* method can be written that makes the internal variable and parameter connections between components and overrides default parameter value and option assignments. Connections are made and broken with *linkTo* and *Unlink* methods of the source parameter or variable. The *linkTo* method will not succeed if the sink parameter or variable is already linked. A bidirectional cascade variable connection is made or broken with a single *linkTo* or *Unlink* method.

Each component composite's *config* method is called before overriding its assignments. This method may be used to instantiate new optional part objects and override their defaults. Each component block or composite is added to the composite's *comp* list in the order in which they are to execute.

A *run* method must be written. This calls in sequence the run method of each component block and composite.

After compilation any number of new composite objects can be created from this composite class in the same way as standard composite objects can be created from its library class.

The following are examples of composite classes: Loop, Ploop, Cascade, TempProcess.

Configuration of Block and Composite Object Instances

The block classes, the composite classes, and the user composite classes can be used to configure as many block and composite object instances as desired without recompiling provided there is a capable configure process.

Examples of block and composite object instances: Loops2 and TempCasc.

Configuration Without Compilation

A control system implemented with PCOs as described above can be configured without compilation. To this end, the following steps can be performed:

Create a configuration object which contains configured PCO objects including parameter values, initial target mode and all connections among various blocks necessary for a closed-loop control.

Download the configuration object to the field device.

Create an object file based on the configuration object.

Start PCO execution from the object file.

Change connections among PCO blocks on the fly by invoking a remote method.

Transfer PCO parameters or variables from a client (host) to a server (field device) together with a string identifying the blocks and related PCO parts and signals .

Add and remove new blocks on the fly. New blocks are instantiated in the device and added into the runList. Connections to the new blocks are made on the fly as well.

Enable PCO optional parts on the fly.

Checkpoint on demand.

Upload an object file into the configurator.

A general RMI RemoteInterface is designed to make all on-line operations listed above possible. The methods in this RemoteInterface provide a useful framework for further development. In particular, several common methods are provided to deal with the following important issues:

Access PCO Parameters and Variables via a string name and an object. The string name follows the naming convention outlined in this document.

Make connections between blocks by invoking a remote method. Sink and source of arrays can be connected by a single remote method call.

Enable any PCO optional parts of any PCO blocks by invoking a single remote method.

The implementation of these methods utilizes the Java Reflection package which is included in Java 1.1.

IEC 1131-3

The IEC 1131-3 specification requires that its Sequential Function Charts and the three different languages namely, Structured Text, Ladder Logic, and Function Blocks work in concert, that is, a user can intermix them and create a control strategy. An example is the creation of a task using Structured Test, which invokes the in-line code execution of a Ladder(s) or a Function Block(s).

Referring to Figure 4 the IEC 1131-3 functionality (Sequential Function Charts, Structured Text, Ladder Logic, and Function Blocks) is shown as a subclass(s) under the UserBlock class. The UserBlock class provides the inputs and outputs as defined by the PCO to all its subclasses, thus enabling interconnection between PCO and IEC 1131-3 objects. The IEC 1131-3 configurator then can produce Java bytecode directly, or can compile source code of another language such as structured text into Java bytecode.

Execution Environment

Process Conrol Objects are preferably run as Java applications (though, they can be run as Java applets, servlets, or the like), with each composite and block object having its own run method, which is called by the run method of the next higher level composite object. Scheduling of a thread is preferably also done in Java, though it can be performed elsewise. For example, a real-time operating system can be used for scheduling, though, it is preferably not hardware specific. The sleep interval until the next run start is

determined by adding the Block Processing Cycle (BPC) time to the present desired run start time and subtracting the current time (in milliseconds).

Each thread preferably has its own object set and run methods. When device is powered up, it starts a PCO application shell. The application shell itself is a Java application. Once it gets started, it runs until device is powered down or rebooted. After the application shell starts, it periodically checks for messages sent from a host running the configurator. In the illustrated embodiment, it understands three types of messages, FileCreation, Commit and Checkpoint.

FileCreation

This message is sent when the *OK* button is pressed. Once this message is received, the application shell serializes the configuration object to a file in the local file system of the device.

Commit

This message is sent when the *Commit* button is pressed. Once this message is received, the application shell does the following:

(1) Load the object file from the file system with the file name given in the message;

(2) Loop through the HashTable built in configuration object to identify all blocks;

(3) Make the block objects available to the remote object;

(4) Start a control thread which executes each object in the runList.

Checkpoint

This message is sent when the *Checkpoint* button is pressed. Once this message is received, the application shell serializes the current configuration object into a file in the local file system. This file is the checkpoint file which can be transferred to any computer via FTP.

Verification and Validation

Verification and validation of control blocks can be implemented in the manner described in copending, commonly assigned United States Patent Application No. 09/345,215, filed June 30, 1999, and its counterpart PCT Application Serial No. US00/16152, filed June 9, 2000, both entitled Process Control and Method with Auto-

Updating, the teachings of which are incorporated herein by reference, and the specification common to both of which is attached in the Appendix hereto.

Platform Communications

Intra-Platform Communication

A cascaded FloatVariable, IntVariable or BoolVariable requires that some of its fields (subobjects) be communicated downstream (forw, alarmSum, uncertainty, maintenance) and others upstream (back, range). All unidirectional variable fields are communicated downstream. Intra-platform communication is done by reference. Variables are stored in an output part of the upstream block and linked to the downstream block with a linkTo() method called at configure time. There is also an unLink() method that can be called to reconfigure.

Objects are not moved from place-to-place within a platform. Input and output signals are instances of the xxVariable class. Variable linking is discussed above. "Constants" such as a gain or time constant are instances of the FloatParameter class. A FloatParameter can be linked to a forw or back parameter of a FloatVariable with a linkTo() method. Each Parameter (such as forw and back) has two status bytes, one is enumerated providing SEVA (clear/blind etc. instead of good/bad) status and cascade status, the other a packed Boolean containing limit bits and linking and setting permission status bits.

Cascade Peer-to-Peer Communication

Block-to-block input-output Variable connections within a station are made by reference. Although the data resides in the upstream block, either block may be the source for a particular attribute. Thus in a cascade (one-to-one) connection, the forw Parameter (value, status, limStatus) and alarmSum are set by the upstream block and the back Parameter and Range are set by the downstream block. The upstream block sets all of the unidirectional Variable attributes.

When the upstream and downstream blocks are in different stations, both unidirectional and bidirectional communication is more complicated. A copy (proxy) of the Variable (or Parameter) that resides in the upstream block must also exist in the downstream block. A cascade Variable must be split into its forward- and back-propagated Parameters. If push (publish-subscribe) communications are used, the source

station needs a publish list of the source object references and a source String names (topics). The sink object needs a subscribe list of the sink object references (proxies) and source object String names (topics).

Figure 17 shows an example of cascade peer-to-peer communication. The cascade control loop consists of AIN1 and PID1 that make up the temperature loop TL1 and AIN2 and PID2 that make up flow loop FL2. TL1 is in device NCAP1 and FL2 is in device NCAP2. In the forward direction from PID1 to PID2, output OUT of PID1 sets the remote setpoint RSP of PID2. In the backward direction from PID2 to PID1, output BACK of PID2 sets the BACK of PID1.

Each device has a Publisher and a Subscriber list and a CommsAPI. Each publisher and subscriber list contains references and topics. The topics are strings indicated in quotes.

Consider the communication in the forward direction from PID1 to PID2. Publisher1 shows reference PID1.OUT, which points to the actual parameter object OUT that needs to be transferred to its subscriber(s) and its corresponding topic "PID1.OUT". This topic is subscibed to by Subscriber2, which shows topic "PID1.OUT" and reference PID1Proxy.RSP. This reference points to the actual parameter proxy object PID1Proxy.RSP where the required parameter object is transferred to. Controller PID2 references this object and uses it when it executes.

Now consider the communication in the back direction from PID2 to PID1. Publisher2 shows reference PID1Proxy.BACK, which points to the actual parameter object that needs to be transferred to its subscriber(s) and its corresponding topic "PID1.BACK" . This topic is subscibed to by Subscriber1, which shows topic "PID1.BACK" and reference PID1.BACK. This reference points to the actual parameter object BACK where the required parameter object is transferred to.

PCO Object Based Transfer

There are two types of communications transfer cascade and unidirectional.

Cascade transfer for peer-to-peer inter-platform communication requires transfer of parameters (value, status, limStatus, time). The previous section provides an example of transfer of parameters for cascade transfer.

Unidirectional transfer for peer-to-peer inter-platform communication requires transfer of variables (forw parameter, range, uncertainty, maintenance, alarmSum).

Object transfer as opposed to fundamental data type allows more information to be transferred in each message, therefore fewer messages and a smaller lookup table. Periodic transfers can be done using publish-subscribe services. One-to-one event driven transfers can be done with client-server services.

Peer-to-peer requires transfer of parameters (value, status, limStatus, time) for cascade transfer or variables (forw parameter, range, uncertainty, maintenance, alarmSum) for unidirectional transfer. Assuming publish-subscribe, each published object must have a unique string name that the subscribing device can recognize.

Operator display requires a trend object (setpoint, measurement, output forw parameters and input and output modes). Publish-subscribe may be used for this also. User setpoint, output, and mode changes can be transmitted using RMI.

Block details require transfer of an object containing the part parameters on each open window each time one changes. Settable changes can be transmitted using RMI.

Configuration requires the streaming of a PCO configuration or checkpoint file. Also it is necessary to be able to invoke remote methods with object arguments.

A historian can receive InfoCollect buffer objects.

Figure 18 provides a summary of the variable and parameter transfer and rules.

Discussion

Blocks structured with a body and one or more input and output parts. Each cascade input part supports local/remote/supervisory switching, setpoint limiting, matched characterization, additive or multiplicative feedforward (bias or ratio setpoint), pv (process variable) output, measurement (in) filtering, deviation 9pv-sp), alarming, and support for bumpless initialization and antiwindup recovery from downstream limiting. Each output part supports manual/auto, supervisory switching, output limiting and alarming, additive or multiplicative feedforward, and bumpless initialization and antiwindup recovery from downstream limiting.

Variables are alarmed at their source and alarm status is communicated to sinks as part of the variable. Range is specified at a measurement source and communicated to sinks as part of the variable. Cascaded range is transmitted upstream. Blocks are the lowest level objects that are recognized at the system level. Each block and its parts must reside in a single station or device. Composites may span stations or devices.

Parts Classes

This section describes the part classes, which are contained in the Parts package. A Part must be contained in a block or other part contained in a block. Each block may have one or more *input* parts, a body, and one or more *output* parts. Input and output Part classes contain or reference either FloatVariable, BoolVariable, or IntVariable signals which enable uni-directional or bi-directional communication between blocks.

An unused optional part has a null reference. The configurator must instantiate an optional part before it can be used or its parameters set.

UniDirFloatInput

Overview

UniDirFloatInput is a measurement path input interface part that initializes and applies charactization, and filtering to its *in* signal.

Attributes

in is a reference to a FloatVariable source. *in.forw.value* is in eng. units.

charac is a reference to an optional signal Characterizer. *charac.xPt[i]* and *charac.yPt[i]* are breakpoint coordinates, float values.

filter is a reference to an optional signal Filter.

options is a packed boolean byte with:

I_O *in* is provided from *device* and *point*

device is the String name of an i/o device that is the source of *in*.

point is an integer used to identify an i/o address within the device.

pv is a Float Variable output representing the processed *in* value.

gain is a FloatParameter that multiplies *pv*.

dtime is a reference to an optional Deadtime part.

Methods for Developer

UniDirFloatInput (constructor)

Arguments: none.

Returns: none.

startup

Arguments: none

tests span.

Returns: true if ready to run.

CascFloatInput

Overview

CascFloatInput is an input interface part that provides initialization, local/remote/supervisory switching, optional setpoint limiting, optional local or supervisory setpoint ramping, optional deviation alarming, and an optional multiplicative or additive feedforward input.

Its internal methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

inHandle() Sets back values and status, processes in.

locRemTarget() Determines the internal target mode.

devHandle(period) Handles setpoint and deviation.

initialize() Provides bumpless mode transitions.

modeHandle() Provides functionality for active mode.

limitStatus(lim) Propagates limit bits to back.status.

Attributes

remSP is an optional reference to the source of the remote setpoint FloatVariable.

supvSP is an optional reference to the source of the supervisory setpoint, a FloatVariable.

lrs is the setpoint local/remote/supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

LOCAL Operator or set method can set *sp*.

REMOTE External block (*remSP*) sets setpoint.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

fdFwd is an optional reference to the source of a Feedforward object.
fdFwd.ff is a reference to the source of a FloatVariable.
fdFwd.gain is a FloatParameter gain applied to *fdFwd.ff*.

ramp is an optional reference to the source of a Ramp object.
ramp.rateDn is a *positive* FloatParameter specifying the rate down.
ramp.rateUp is a *positive* FloatParameter specifying the rate up.
ramp.target is a FloatParameter that contains the termination value.
ramp.trigger is a BoolParameter that starts the ramp.

limit is an optional reference to the source of a setpoint Limit object.
limit.hi is the high limit, FloatParameter
limit.lo is the low limit, FloatParameter

alarmDevi is an optional reference to the source of a *deviation* Alarm object.
alarmDev.hi is the high alarm limit, a FloatParameter.
alarmDev.lo is the low alarm limit, a FloatParameter.
alarmDev.hys is the alarm hysteresis, a FloatParameter.
alarmDev.delay is an optional alarm delay in seconds, a FloatParameter
alarmDev.noAck is a boolean, alarm requires no acknowledge if true.

sp is a FloatVariable representing the current setpoint. In the LOCAL mode *sp* can be set

deviation is a FloatVariable output representing *in* - *sp*. It can be alarmed.

Attributes from super class.

in is a reference to a FloatVariable source. *in.forw.value* is in eng. units.

charac is a reference to an optional signal Characterizer.
charac.xPt[i] and *charac.yPt[i]* are breakpoint coordinates, float values.

filter is a reference to an optional signal Filter.

options is a packed boolean byte with bits assigned as:
SP_PV_TRK_MAN Setpoint tracks PV when output is manually set.
SP_DN_UNCER Setpoint is decreased by the uncertainty.
SP_UP_UNCER Setpoint is increased by the uncertainty.

MULT_FF Feedforward is multiplicative, else it is additive.

device is the String name of an i/o device that is the source of *in*.

point is an integer used to identify an i/o address within the device.

pv is a Float Variable output representing the processed *in* value.

Methods for Developer

CascFloatInput (constructor)

Arguments: none.

Instantiates *sp* and *deviation*.

Returns: none.

startup

Arguments: none.

Called when out-of-service.

Returns: true if ready to run.

inHandle

Arguments: none.

Sets back values and status, processes *in* and feedforward.

Returns: none.

locRemTarget

Arguments: none.

Determines the internal target mode.

Returns: none.

devHandl

Arguments: *period* is the time since the last update in seconds, a double.

Handles setpoint and deviation.

Returns: none

initialize

Arguments: none.

Provides bumpless mode transitions

Returns: none.

modeHandle

Arguments: none.

Provides functionality for active mode.

Returns: none.

limitStatus

Arguments: *lim* holds limit bits to be propagated upstream, an integer.

Propagates limit bits to back.status.

Returns: none.

UniDirFloatOutput

Overview

UniDirFloatOutput is an output interface part that provides auto/manual/supervisory switching, optional out limiting, optional out alarming, and an optional multiplicative or additive feedforward input.

Its methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

manAutoTarget() Determines the internal target mode.

modeHandle(period) Provides functionality for active mode.

Attributes

fdFwd is an optional reference to the source of a Feedforward object.
fdFwd.ff is a reference to the source of a FloatVariable.
wd.gain is a FloatParameter gain applied to *fdFwd.ff*.

supvOut is an optional reference to the source of the supervisory output, a FloatVariable.

mas is the output manual-auto-supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

MANUAL Operator or set method can set *out*.

AUTO Block algorithm sets *out*.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

INTERLOCK *interlock.in* sets *out* when *interlock.trigger* is true.

options is a packed boolean byte with bits assigned as:

LIM_MAN Limits are applied to manual output.

ALM_MAN Alarming is applied to a manual output.

MAN_START Block returns from out-of-service in MANUAL mode.

MULT_FF Feedforward is multiplicative, else it is additive.

ALARM_BLURRED Block alarm triggered by BLURRED signal status.

I_O *out* is provided to *device* and *point*.

limit is an optional reference to the source of an *out* Limit object.

limit.hi is the high limit, FloatParameter

limit.lo is the low limit, FloatParameter

alarmHi is an optional reference to the source of an *out* Alarm object.

alarmHi.hi is the high alarm limit, a FloatParameter.

alarmHi.lo is the low alarm limit, a FloatParameter.

alarmHi.hys is the alarm hysteresis, a FloatParameter.

alarmHi.delay is an optional alarm delay in seconds, a FloatParameter

alarmHi.noAck is a boolean, alarm requires no acknowledge if true.

alarmHiHi is an optional reference to the source of an *out* Alarm object.

alarmHiHi.hi is the high alarm limit, a FloatParameter.

alarmHiHi.lo is the low alarm limit, a FloatParameter.

alarmHiHi.hys is the alarm hysteresis, a FloatParameter.

alarmHiHi.delay is an optional alarm delay in seconds, a FloatParameter

alarmHiHi.noAck is a boolean, alarm requires no acknowledge if true.

interlock is an optional object used to invoke an override of the current Mode.

interlock.trigger is a BoolParameter used to invoke INTERLOCK Mode.

interlock.reset is a BoolParameter used to return to normal Mode, provided *interlock.trigger* is false.

interlock.in is a FloatParameter that sets *out* in the INTERLOCK Mode.

device is the String name of an i/o device that is the sink of *out.*

point is an integer used to identify an i/o address within the device.

out is a FloatVariable representing the current output. In the MANUAL mode *out* can be set

fbk is the integral feedback input, a FloatParameter, which is connected by default to *out.back* if it is connected externally, otherwise to *out.forw.*.

Methods for Developer

UniDirFloatOutput (constructor)

Arguments: none.

Instantiates *out.*

Returns: none.

startup

Arguments: none.

Attributes from super class

FdFwd is an optional reference to the source of a Feedforward object. *fdFwd.ff* is a reference to the source of a FloatVariable. *fdFwd.gain* is a FloatParameter gain applied to *fdFwd.ff.*

supvOut is an optional reference to the source of the supervisory output, a FloatVariable.

mas is the output manual-auto-supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

MANUAL Operator or set method can set *out.*

AUTO Block algorithm sets *out.*

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

INTERLOCK *interlock.in* sets *out* when *interlock.trigger* is true.

options is a packed boolean byte with bits assigned as:

LIM_MAN Limits are applied to manual output.

ALM_MAN Alarming is applied to a manual output.

MAN_START Block returns from out-of-service in MANUAL mode.

MULT_FF Feedforward is multiplicative, else it is additive.

ALARM_BLURRED Block alarm triggered by BLURRED signal status.

limit is a reference to the source of an *out* Limit object.

limit.hi is the high limit, FloatParameter

limit.lo is the low limit, FloatParameter

alarmHi is an optional reference to the source of an *out* Alarm object.

alarmHiHi is an optional reference to the source of an *out* Alarm object.

alarmHiHi.hi is the high alarm limit, a FloatParameter.

alarmHiHi.lo is the low alarm limit, a FloatParameter.

alarmHiHi.hys is the alarm hysteresis, a FloatParameter.

alarmHiHi.delay is an optional alarm delay in seconds, a FloatParameter

alarmHiHi.noAck is a boolean, alarm requires no acknowledge if true.

interlock is an optional object used to invoke an override of the current Mode.

interlock.trigger is a BoolParameter used to invoke INTERLOCK Mode.

interlock.reset is a BoolParameter used to return to normal Mode, provided *interlock.trigger* is false.

interlock.in is a FloatParameter that sets *out* in the INTERLOCK Mode.

device is the String name of an i/o device that is the sink of *out*.

point is an integer used to identify an i/o address within the device.

out is a FloatVariable representing the current output. In the MANUAL mode *out* can be set

Methods for Developer

CascFloatOutput (constructor)

Arguments: none.

Instantiates *out* and *limit.*

Returns: none.

manAutoTarget

Arguments: none.

Determines the internal target mode

Returns: none.

bkCalc

Arguments: none.

Compensates integral feedback for feedforwards.

Returns: none.

initialize

Arguments: none.

Provides bumpless mode transitions.

Returns: none.

Methods from super class

startup

Arguments: none.

Called when out-of-service, returns true if ready to run.

Returns: none.

modeHandle

Arguments: *period* is the time since the last update in seconds, a double.

Provides functionality for active mode.

Returns: none.

Alarm

Overview

Alarm is an optional part for all outputs. Its float method is *func(x,type,period)* where *type* is true for HH and LL, false for HI and LO. Its boolean method is *func(x,period)*. Alarm reporting can be delayed if the optional *delay* parameter is not null or inhibited if *inhib* is true.

Attributes

hi is the high alarm limit, a FloatParameter.

lo is the low alarm limit, a FloatParameter.

hys is the alarm hysteresis, a FloatParameter.

lim is the boolean alarm state, a BoolParameter.

lim is an optional output, true if alarm is active, a BoolParameter.

delay is an optional alarm delay in seconds, a FloatParameter.

noAck is a boolean, alarm requires no acknowledge if true.

inhib is a boolean, inhibits alarm if true.

Methods for Developer

Alarm (constructor)

Arguments: none,. signifying a float alarm.

Returns: none.

Alarm (constructor)

Arguments: *r* is a FloatRange.object used to set the alarm limits.

Returns: none.

Alarm (constructor)

Arguments: b is a boolean signifying a boolean alarm..

Returns: none.

func

Arguments: *x* is the alarmed BoolVariable.

period is the time since the last update in seconds, a double *blurAl*

Returns: none.

func

Arguments: *x* is the alarmed BoolVariable.

period is the time since the last update in seconds, a double *blurAl*

Returns: none.

Characterizer

Overview

Characterizer is an optional part class used in the CascFloatInput part, UniDirFloatInput part, LinearFlow part, and ProcessSim block classes. Its method is $y = func(x)$. The number of breakpoints is specified as the constructor argument.

Attributes

xPt[i] and *yPt[i]* are breakpoint coordinates, float values.

Methods for Developer

Characterizer (constructor)

Arguments: *nPt* is an int number of (*xPt, yPt*) break points.

Returns: none.

Func

Arguments: *u*. is the input to the function, a double (*x* if *forw* is true).

Forw is true *if* $u = x, v = y$, else $u = y, v = x$.

Returns: *v* is the output of the function, a double (*y* if *forw* is true)

FeedForward

Overview

Feedforward is an optional part class used in CascFloatInput, CascFloatOutput, and UniDirFloatOutput classes. It provides multiplicative or additive compensation for the input. Its methods are $f = forwFunc(f, mult, out)$ and $b = backFunc(b,mult)$, where *mult* and *out* are booleans indicating if true that the compensation is multiplicative and applied to output respectively.

Attributes

ff is a reference to the source of a FloatVariable.

Gain is a FloatParameter gain applied to *ff*.

Methods for Developer

Feedforward (constructor)

Arguments: none.

Returns: none.

ForwFunc

Arguments: *sig* is the internal pre feedforward *out.forw.value*, a double.

mult is true if feedforward is multiplicative.

out is true if the feedforward is an output part. Then a multiplicative feedforward is normalized with its high-range value.

Returns: the prelimited *out.forw.value*, a double.

backFunc

Arguments: *sig* = *fbk.value*, a double.

mult is true if feedforward is multiplicative.

Returns: the back-calculated internal fbk, a double.

Filter

Overview

Filter is a part class used in the CascFloatInput part, UniDirFloatInput part, and ProcessSim block classes. Its method is *y* = *func(x,period)*. It is initialized with *init(x)*.

$$s = \frac{d()}{dt}$$

$$y = \frac{x}{1 + fTime \cdot s + .fType \cdot (fTime \cdot s)^2}$$

$$v = fTime \cdot s \cdot y$$

$$returns: \frac{(1 + lead \cdot s) \cdot x}{1 + fTime \cdot s + .fType \cdot (fTime \cdot s)^2}$$

Attributes

FTime is the filter time in seconds, a FloatParameter.

lead is the lead time in seconds, a FloatParameter.

fType specifies the filter denomiator type, a FloatParameter.

1/2 Butterworth, 1/3 Averaging, 1/4 Two-Lag, 0 One-Lag

y is the low-pass filtered output, a double.

v is the band-pass filtered output, a double.

Methods for Developer

Filter (constructor)

Arguments: none.

Instantiates *fTime* to 0., *fType* to .5, and *lead* to 0.

Returns: none.

init

Arguments: *x* is the filter input, a double

Initializes the filter output to equal its input..

Returns: none.

func

Arguments: *x* is the filter input, a double.

period is the time since the last update in seconds, a double

Returns: the filter output, a double.

Limit

Overview

Limit is a part class applied to the setpoint in the CascFloatInput, and to the *out* of a CascFloatOutput, and UniDirFloatOutput classes. Its method is $x = func(x)$. The high limit cannot be less than the low limit. Neither limit may be more than 2% beyond the range limits.

Attributes

hi is a reference to a FloatVariable containing the high limit.

lo is a reference to a FloatVariable containing the low limit.

Methods for Developer

Limit (constructor)

Arguments: none.

Instantiates *hi* and *lo*.

Returns: none.

func

Arguments: *x* is the FloatVariable to be limited.

Returns: none.

Ramp

Overview

Ramp is an optional part class applied to the local setpoint in the CascFloatInput class. It provides a triggered ramp function with the method $y = func(x,period)$.

Attributes

rateDn is a *positive* FloatParameter specifying the rate down.

rateUp is a *positive* FloatParameter specifying the rate up.

target is a FloatParameter that contains the termination value.

trigger is a BoolParameter that starts the ramp.

Methods for Developer

Ramp(constructor) (boolean link)

Arguments: none.

Returns: none.

Ramp(constructor)

Arguments: none.

Returns: none.

func

Arguments: spt is the input to the ramp, a double.
period is the time since the last update in seconds, a double.

Returns: the ramp result, a double.

UniDirBoolInput

Overview

UniDirBoolInput is a measurement path input interface part that initializes to its *in* signal.

Attributes

in is a reference to a BoolVariable source.

filter is a reference to an optional signal Filter.

options is a packed boolean byte with:
I_O *in* is provided from *device* and *point*

device is the String name of an I/O device that is the source of *in*.

point is an integer used to identify an I/O address within the device.

pv is a BoolVariable output representing the processed *in* value.

Methods for Developer

UniDirFloatInput (constructor)

Arguments: none.

Returns: none.

startup

Arguments: none

Returns: true if ready to run.

CascBoolInput

Overview

CascBoolInput is an input interface part that provides initialization and local/remote/supervisory switching.

Its internal methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

inHandle() Sets back values and status, processes in.

locRemTarget() Determines the internal target mode.

devHandle(period) Handles setpoint and deviation.

initialize() Provides bumpless mode transitions.

modeHandle() Provides functionality for active mode.

Attributes

remSP is an optional reference to the source of the remote setpoint BoolVariable.

supvSP is an optional reference to the source of the supervisory setpoint, a BoolVariable.

lrs is the setpoint local/remote/supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

LOCAL Operator or set method can set *sp*.

REMOTE External block (*remSP*) sets setpoint.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

sp is a BoolVariable representing the current setpoint. In the LOCAL mode *sp* can be set

filter is a reference to an optional signal BoolFilter.

options is a packed boolean byte with bits assigned as:

pv is a IntVariable output representing the processed *in* value.

Methods for Developer

CascBoolInput(constructor)

Arguments: none.

Returns: none.

startup

Arguments: none.

Called when out-of-service.

Returns: true if ready to run.

inHandle

Arguments: none.

Sets back values and status.

Returns: none.

locRemTarget

Arguments: none.

Determines the internal target mode.

Returns: none.

devHandl

Arguments: none.

Handles setpoint.

Returns: none

initialize

Arguments: none.

Provides bumpless mode transitions

Returns: none.

modeHandle

Arguments: none.

Provides functionality for active mode.

Returns: none.

UniDirBoolOutput

Overview

UniDirBoolOutput is an output interface part that provides auto/manual/supervisory switching.

Its methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

manAutoTarget() Determines the internal target mode.

modeHandle(period) Provides functionality for active mode.

Attributes

supvOut is an optional reference to the source of the supervisory output, a BoolVariable.

mas is the output manual-auto-supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

MANUAL Operator or set method can set *out*.

AUTO Block algorithm sets *out*.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

INTERLOCK *interlock.in* sets *out* when *interlock.trigger* is true.

options is a packed boolean byte with bits assigned as:

ALARM_BLURRED Block alarm triggered by BLURRED signal status.

I_O *out* is provided to *device* and *point*.

alarm is an optional reference to the source of an *out* Alarm object.

alarmHi.hi is the high alarm limit, a FloatParameter.

alarmHi.lo is the low alarm limit, a FloatParameter.

alarmHi.hys is the alarm hysteresis, a FloatParameter.

alarmHi.delay is an optional alarm delay in seconds, a FloatParameter

alarmHi.noAck is a boolean, alarm requires no acknowledge if true.

interlock is an optional object used to invoke an override of the current Mode.

interlock.trigger is a BoolParameter used to invoke INTERLOCK Mode.

interlock.reset is a BoolParameter used to return to normal Mode, provided *interlock.trigger* is false. *interlock.in* is a FloatParameter that sets *out* in the INTERLOCK Mode.

device is the String name of an I/O device that is the sink of *out*.

point is an integer used to identify an I/O address within the device.

out is a BoolVariable representing the current output. In the MANUAL mode *out* can be set.

Methods for Developer

UniDirBoolOutput (constructor)

Arguments: none.

Instantiates *out*.

Returns: none.

startup

Arguments: none..

Called when out-of-service, returns true if ready to run.

Returns: none.

manAutoTarget

Arguments: none.

Determines the internal target mode

Returns: none.

modeHandle

Arguments: *period* is the time since the last update in seconds, a double.

Provides functionality for active mode.

Returns: none.

CascBoolOutput

Overview

CascBoolOutput is an output interface part that provides initialization, auto/manual/supervisory switching.

Its methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

manAutoTarget() Determines the internal target mode.

bkCalc() Set the backword values.

initialize() Provides bumpless mode transitions.

modeHandle(period) Provides functionality for active mode.

Attributes

out0 is a BoolVariable representing the first output. In the MANUAL mode *out0* can be set.

out1 is a BoolVariable representing the second output. In the MANUAL mode *out1* can be set.

point0 is an integer used to identify the I/O address of *out0*.

point1 is an integer used to identify the I/O address of *out1*.

Attributes from super class

supvOut is an optional reference to the source of the supervisory output, a BoolVariable.

mas is the output manual-auto-supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

MANUAL Operator or set method can set *out*.

AUTO Block algorithm sets *out*.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

INTERLOCK *interlock.in* sets *out* when *interlock.trigger* is true.

alarm is an optional reference to the source of an *out* Alarm object.

alarmHi.hi is the high alarm limit, a FloatParameter.

alarmHi.lo is the low alarm limit, a FloatParameter.

alarmHi.hys is the alarm hysteresis, a FloatParameter.

alarmHi.delay is an optional alarm delay in seconds, a FloatParameter

alarmHi.noAck is a boolean, alarm requires no acknowledge if true.

interlock is an optional object used to invoke an override of the current Mode.

interlock.trigger is a BoolParameter used to invoke INTERLOCK Mode. *interlock.reset* is a BoolParameter used to return to normal Mode, provided *interlock.trigger* is false. *interlock.in* is a FloatParameter that sets *out* in the INTERLOCK Mode.

device is the String name of an I/O device that is the sink of *out.*

point is an integer used to identify an I/O address within the device.

out is a BoolVariable. It is assigned to *out0*, the first output.

Methods for Developer

CascBoolOutput (constructor)

Arguments: none.

Instantiates *out0* and *out1*.

Returns: none.

manAutoTarget

Arguments: none.

Determines the internal target mode

Returns: none.

bkCalc

Arguments: none.

Set the backword values for *out0* and *out1*.

Returns: none.

initialize

Arguments: none.

Provides bumpless mode transitions.

Returns: none.

Methods from super class

startup

Arguments: none.

Called when out-of-service, returns true if ready to run.

Returns: none.

modeHandle

Arguments: *period* is the time since the last update in seconds, a double.

Provides functionality for active mode.

Returns: none.

BoolFilter

Overview

BoolFilter is a part class used in classes such as DeviceControl and Logic. Its method is $y = func(x, period)$. It is initialized with *init()* to $y = 0$ each time the input changes state.

$$s = \frac{d()}{dt}$$

$$y = \frac{1}{1 + fTime \cdot s}$$

$$returns: \begin{cases} tinput\ state & y \geq 0.632 \\ output\ state & otherwise \end{cases}$$

Attributes

fTime is the filter time in seconds, a FloatParameter.

Methods for Developer

BoolFilter(constructor)

Arguments: none.

Returns: none.

init

Arguments: x is the filter input, a double

Initializes the filter output to its input.

Returns: none.

func

Arguments: x is the filter input, a double.

period is the time since the last update in seconds, a double

Returns: the filter output, a boolean.

UniDirIntInput

Overview

UniDirIntInput is a measurement path input interface part that initializes to its *in* signal.

Attributes

in is a reference to a IntVariable source.

filter is a reference to an optional signal IntFilter.

options is a packed boolean byte with:

I_O *in* is provided from *device* and *point*

device is the String name of an I/O device that is the source of *in*.

point is an integer used to identify an I/O address within the device.

pv is a IntVariable output representing the processed *in* value.

Methods for Developer

UniDirIntInput (constructor)

Arguments: none.

Returns: none.

startup

Arguments: none

Returns: true if ready to run.

CascIntInput

Overview

CascIntInput is an input interface part that provides initialization and local/remote/supervisory switching.

Its internal methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

inHandle() Sets back values and status, processes in.

locRemTarget() Determines the internal target mode.

devHandle(period) Handles setpoint and deviation.

initialize() Provides bumpless mode transitions.

modeHandle() Provides functionality for active mode.

Attributes

remSP is an optional reference to the source of the remote setpoint IntVariable.

supvSP is an optional reference to the source of the supervisory setpoint, a IntVariable.

lrs is the setpoint local/remote/supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

LOCAL Operator or set method can set *sp*.

REMOTE External block (*remSP*) sets setpoint.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

sp is a IntVariable representing the current setpoint. In the LOCAL mode *sp* can be set

pv is a IntVariable output representing the processed *in* value.

Methods for Developer

CascBoolInput(constructor)

Arguments: none.

Returns: none.

startup

Arguments: none.

Called when out-of-service.

Returns: true if ready to run.

inHandle

Arguments: none.

Sets back values and status.

Returns: none.

locRemTarget

Arguments: none.

Determines the internal target mode.

Returns: none.

devHandl

Arguments: none.

Handles setpoint.

Returns: none

initialize

Arguments: none.

Provides bumpless mode transitions

Returns: none.

modeHandle

Arguments: none.

Provides functionality for active mode.

Returns: none.

UniDirIntOutput

Overview

UniDirIntOutput is an output interface part that provides auto/manual/supervisory switching.

Its methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

manAutoTarget() Determines the internal target mode.

modeHandle(period) Provides functionality for active mode.

Attributes

supvOut is an optional reference to the source of the supervisory output, a IntVariable.

mas is the output manual-auto-supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

MANUAL Operator or set method can set *out*.

AUTO Block algorithm sets *out*.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

INTERLOCK *interlock.in* sets *out* when *interlock.trigger* is true.

options is a packed boolean byte with bits assigned as:

MAN_START Block returns from out-of-service in MANUAL mode.

ALARM_BLURRED Block alarm triggered by BLURRED signal status.

I_O *out* is provided to *device* and *point*.

alarm is an optional reference to the source of an *out* Alarm object.

alarmHi.hi is the high alarm limit, a FloatParameter.

alarmHi.lo is the low alarm limit, a FloatParameter.

alarmHi.hys is the alarm hysteresis, a FloatParameter.

alarmHi.delay is an optional alarm delay in seconds, a FloatParameter

alarmHi.noAck is a boolean, alarm requires no acknowledge if true.

interlock is an optional object used to invoke an override of the current Mode.

interlock.trigger is a BoolParameter used to invoke INTERLOCK Mode. *interlock.reset* is a BoolParameter used to return to normal Mode, provided *interlock.trigger* is false. *interlock.in* is a FloatParameter that sets *out* in the INTERLOCK Mode.

device is the String name of an I/O device that is the sink of *out*.

point is an integer used to identify an i/o address within the device.

out is a IntVariable representing the current output. In the MANUAL mode *out* can be set.

Methods for Developer

UniDirBoolOutput (constructor)

Arguments: none.

Instantiates *out*.

Returns: none.

startup

Arguments: none..

Called when out-of-service, returns true if ready to run.

Returns: none.

manAutoTarget

Arguments: none.

Determines the internal target mode

Returns: none.

modeHandle

Arguments: *period* is the time since the last update in seconds, a double.

Provides functionality for active mode.

Returns: none.

CascIntOutput

Overview

CascIntOutput is an output interface part that provides initialization and auto/manual/supervisory switching.

Its methods are called by its container block.

startup() Called when out-of-service, returns true if ready to run.

manAutoTarget() Determines the internal target mode.

bkCalc() Set the backword values.

initialize() Provides bumpless mode transitions.

modeHandle(period) Provides functionality for active mode.

Attributes

point is an integer used to identify address of *out*.

Attributes from super class

supvOut is an optional reference to the source of the supervisory output, a IntVariable.

mas is the output manual-auto-supervisory Mode having target, actual, permitted, and last packed boolean bytes whose bits are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

MANUAL Operator or set method can set *out*.

AUTO Block algorithm sets *out*.

SUPER Supervisory task (*supvSP*) sets setpoint.

LINKED Permitted attribute indicates target connected to external source.

INTERLOCK *interlock.in* sets *out* when *interlock.trigger* is true.

alarm is an optional reference to the source of an *out* Alarm object.

alarmDev.hi is the high alarm limit, a FloatParameter.

alarmDev.lo is the low alarm limit, a FloatParameter.

alarmDev.hys is the alarm hysteresis, a FloatParameter.

alarmDev.delay is an optional alarm delay in seconds, a FloatParameter

alarmDev.noAck is a boolean, alarm requires no acknowledge if true.

interlock is an optional object used to invoke an override of the current Mode.

interlock.trigger is a BoolParameter used to invoke INTERLOCK Mode. *interlock.reset* is a BoolParameter used to return to normal Mode, provided *interlock.trigger* is false. *interlock.in* is a FloatParameter that sets *out* in the INTERLOCK Mode.

device is the String name of an I/O device that is the sink of *out*.

point is an integer used to identify an I/O address within the device.

out is a IntVariable representing the current output. In the MANUAL mode *out* can be set.

Methods for Developer

CascBoolOutput (constructor)

Arguments: none.

Instantiates *out*.

Returns: none.

manAutoTarget

Arguments: none.

Determines the internal target mode

Returns: none.

bkCalc

Arguments: none.

Set the backword values for *out*.

Returns: none.

initialize

Arguments: none.

Provides bumpless mode transitions.

Returns: none.

Methods from super class

startup

Arguments: none.

Called when out-of-service, returns true if ready to run.

Returns: none.

modeHandle

Arguments: *period* is the time since the last update in seconds, a double.

Provides functionality for active mode.

Returns: none.

IntFilter

Overview

IntFilter is a part class used in the DiscreteInput class. Its method is $y =$ *func(x,period)*. It is initialized with *init(to y = 1 each time the input changes state.*

$$s = \frac{d()}{dt}$$

$$y = \frac{1}{1 + fTime \cdot s}$$

$$returns: \begin{cases} input\ state & y \geq 0.632 \\ output\ state & otherwise \end{cases}$$

Attributes

fTime is the filter time in seconds, a FloatParameter.

Methods for Developer

BoolFilter(constructor)

Arguments: none.

Returns: none.

init

Arguments: *x* is the filter input, a double

Initializes the filter output to equal its input..

Returns: none.

func

Arguments: *x* is the filter input, a double.

period is the time since the last update in seconds, a double

Returns: the filter output, a boolean.

PIDParameters

Overview

PIDParameters is an optional part class containing tuning parameters used in the AdvancedPID class.

Attributes

PGain is the nondimensional prpoortional gain, a FloatParameter.

integral is the integral time in seconds, a FloatParameter.

derivative is the derivative time in seconds, a FloatParameter.

relGain is the nondimensional relative gain on setpoint, a FloatParameter.

dtime is the optional deadtime in seconds, a FloatParameter

fTime is the measurement filter time in seconds, a FloatParameter.

bypass is an optional BoolParameter, if true the controller is bypassed.

zone is an optional object from the Zone class, used to position a region of low gain where the control error is small.

sampleReady is a BoolParameter indicating that a new measurement value is available.

Its default value is true.

options is a packed-boolean byte with the following assignments:

IFS_IF_BAD_RSP	initiates failsafe if *remoteSP* is BLIND
IFS_IF_BAD_IN	initiates failsafe if *in* is BLIND .
INCOPT	increasing *in* causes *out* to increase.
PD	no controller integral action.
I_ONLY	only controller integral action.
PASS_RANGE	the range is passed to the setpoint.

Methods for Developer

PIDParameters (constructor)

Arguments: none.

Instantiates *pGain, integral, derivative,* and *relGain.*

Returns: none.

Deadtime

Overview

Deadtime is a part class used in the AdvancedPID and ProcessSim classes. Its method y = func(x,period) implements a bucket-brigade delay line. The number of buckets is specified as the constructor argument.

Attributes

dt is the deadtime in seconds, a FloatParameter.

Methods for Developer

Deadtime() (constructor)

Arguments: none.

Instantiates *dt, uses 10 buckets.*

Returns: none.

Deadtime (constructor)

Arguments: numBuckets is the number of buckets, an int..

Instantiates *dt.*

Returns: none.

hold

Arguments: none

Returns: none.

init

Arguments: *x*

Initializes the delayed signal to equal input *x*.

Returns: none.

func

Arguments: *x*

period is the time since the last update in seconds, a double.

Returns: the value of the delayed signal, a double..

Zone

Overview

Zone is a part class used in the AdvancedPID class. It is a three segment characterizer used as a nonlinear noise filter and in pH control. Its method is $y = func(x)$.

Attributes

hi is the upper control error breakpoint in percent of *in* span, a FloatParameter.

lo is the lower control error breakpoint in percent of *in* span, a FloatParameter.

Methods for Developer

Zone (constructor)

Arguments: none.

Instantiates *hi*, *lo*, and *k*.

Returns: none.

func

Arguments: *x* is the input signal, a double.

Returns: the Zone output signal, a double.

FailSafe

Overview

FailSafe is a part class used by AnalogOutput and Logic classes.

Attributes

fsTime *fsTimeis a* FloatParameter wait time in seconds before failsafe is activated.

var *var* is the FloatParameter that sets the valve target when failsafe is active.

bool is the value of a BoolParameter that sets *out* when failsafe is active.

Methods for Developer

Failsafe (constructor)

Arguments: none.

Returns: none.

LinearFlow

Overview

LinearFlow is an optional part used to cause the *out* Variable of an AnalogOutput block to be a valve position target that will make valve flow linear with the AnalogOutput setpoint. Its methods

are x = forw(v) and v = back(x). The parameter 0 <= beta <= 1 is defined as:

Valve pressures at max opening: p_1, p_2

Supply pressures: p_s, p_a

Liquid: $\beta = \frac{p_1 - p_2}{p_s - p_a}$

Gas: $\beta = \frac{p_1^2 - p_2^2}{p_s^2 - p_a^2}$

0 <= v <= 1 is normalized flow

0 <= y <= 1 is normalized valve flow area

0 <= x <= 1 is normalized valve stem position.

Maximum liquid power is delivered to the load when beta is 1/3 and v = 1. Then the ratio of max-to-min slopes dv/dy is 5.2. The pressure drop correction is:

$$y^2 = \frac{\beta \cdot v^2}{1 - v^2 \cdot (1 - \beta)}$$

x = func(y,true)

Attributes

beta is the pressure drop ratio, a FloatParameter.

turndown is (max flow area)/(min flow area) for an equql-percentage valve, an optional FloatParameter.

charac is an optional object of the Characterizer class providing normalized stroke vs. normalized area in the forward direction.

Methods for Developer

LinearFlow (constructor)

Arguments: none.

Returns: none.

forw

Arguments: the normalized target flow, a double.

Returns: the nomalized target valve position, a double.

back

Arguments: the nomalized actual valve position, a double.

Returns: the normalized back-calculated flow, a double.

InfoBuffer

Overview

InfoBuffer is a part class used by the InfoCollect class. Its method *func* implements double buffering with a bucket-brigade delay line for each parameter stored. The number of buckets (time steps) is specified as a constructor argument.

Attributes

xTime[buf][step] is an array of time stamp in milliseconds since Jan 1, 1970, a long.

xFloat[buf][step][nFP] is an optional array of FloatParameter *value* floats.

xFStatus[buf][step][nFP] is an optional array of FloatParameter *status* bytes.

xBool[buf][step][nBP] is an optional array of BoolParameter *values*.

xBStatus[buf][step][nBP] is an optional array of BoolParameter *status* bytes.

xMode[buf][step][nM] is an optional array of Mode *actual* bytes.

xAlarm[buf][step][nAl] is an optional array of AlarmSum shorts.

Methods for Developer

InfoBuffer(constructor)

Arguments: *x* is the time stamp, a long.

NumSteps is the length of the buffer, an int.

Returns: none.

InfoBuffer(constructor)

Arguments: *x* is an array of FloatParameters.

nP is the number of FloatParameters

NumSteps is the length of the buffer, an int.

Returns: none.

InfoBuffer(constructor)

Arguments: *x* is an array of BoolParameters.

nP is the number of BoolParameters

NumSteps is the length of the buffer, an int.

Returns: none.

InfoBuffer(constructor)

Arguments: *x* is an array of Modes.

nP is the number of Modes.

NumSteps is the length of the buffer, an int.

Returns: none.

InfoBuffer(constructor)

Arguments: *x* is an array of bytes.

nP is the number of bytes

NumSteps is the length of the buffer, an int.

Returns: none.

func

Arguments: buf is number of the buffer, an int.

step is the position in the buffer, an int.

x is the time stamp, a long.

Returns: none.

func

Arguments: buf is number of the buffer, an int.

step is the position in the buffer, an int.

x is an array of FloatParameters.

Returns: none.

func

Arguments: buf is number of the buffer, an int.

step is the position in the buffer, an int.

x is an array of BoolParameters.

Returns: none.

func

Arguments: buf is number of the buffer, an int.

step is the position in the buffer, an int.

x is an array of Modes.

Returns: none.

func

Arguments: buf is number of the buffer, an int.
step is the position in the buffer, an int.
x is an array of shorts.

Returns: none.

TimeCount

Overview

TimeCount is a part class that performs a timer function. Method *func* returns true when time reaches pre-specified time value.

Attributes

tTime	is the time in seconds, a FloatParameter.
timeElapsed	is the time elapsed since timer starts, a double.
statusFlag	indicates whether timer is running or stopped.

Methods for Developer

TimeCount (constructor)

Arguments: *countUp*, a boolean. Timer counts up from 0 if *countUp* is true; counts down from *tTime* otherwise.

Instantiates *tTime*, a FloatParameter.

Returns: none.

init

Arguments: none.

Initializes the timer. It should be called before timer starts.

Returns: none.

func

Arguments: *period* is the time since the last update in seconds, a double.

Updates the timer.

Returns: none.

stop

Arguments: none.

Stops the timer. It should be called when the timer is going to stop.

Returns: none.

getTime

Arguments: none.

Gets the time elapsed since timer starts.

Returns: elapsed time, a double.

Interlock

Overview

Interlock is an optional part class used to invoke an override of the current *out* Variable.

Attributes

interlock.trigger is a BoolParameter used to invoke INTERLOCK Mode.

interlock.reset is a BoolParameter used to return to normal Mode.

interlock.in is a FloatParameter or BoolParameter that sets *out in the INTERLOCK Mode.*

Methods for Developer

None.

Discussion of Parts Classes

Parts are optional or mandatory parts of blocks, providing optional functionality or functionality shared by many blocks. Generally, each block has one or more *input* parts, a body, and one or more *output* parts. Exceptions are InfoCollect, Event, and IOInterface blocks.

input (-Input) and *output* (-Output) parts may be of the unidirectional (UniDir-) or cascade (Casc-) types and may be for float (-Float-), boolean (-Bool-), or short (16 bit) integer (-Int-) value data types.

Unidirectional inputs and outputs are used in blocks, such as AnalogInput and ProcessSim, located in a measurement signal path. Dynamic elements in these blocks are initialized to steady-state corresponding to the current *in* value.

A unidirectional or cascade output part provides manual, auto, supervisory mode switching to select the source of the *out* variable. Before the mode switch the auto signal may be modified by an additive or multiplicative feedforward (unidirectional) source. After the mode switch the output is limited (optional in manual) and may be alarmed

(optional in manual). In a cascade output part the *out* limits are each bidirectional variables that can be connected from a bidirectional *out* variable of another AdvancedPID block to implement a constraint or override. Each output part has an *interlock* input that when triggered overrides the out variable from other sources.

A cascade input part provides local/remote/supervisory mode switching to select the setpoint *sp* (bidirectional) source. The local setpoint may be ramped, the setpoint limited and modified by an additive or multiplicative feedforward (unidirectional) source. If the cascade part has an *in* (unidirectional measurement) signal, both it and the signal in the setpoint path may be linearized with matched functions before being passed to the block body. Then the signal in the measurement path is filtered (quadratic Butterworth).The cascade input part has two unidirectional output variables, *pv* and *deviation*. When the block has an *in*, as in an AdvancedPID block, it is passed to the *pv*. When it doesn't, as in an AnalogOutput block, *pv* is back calculated from a signal passed from the body. In either case *deviation*, which may be alarmed, is the difference between *pv* and *sp*.

Signals Classes

This section describes the signal classes, which are contained in the Signals package. Objects from Signals classes contain the information communicated between linked blocks or inserted into blocks as constants. These classes also provide methods to be used by users and tasks for setting and getting values and for linking and unlinking parameters and variables. The Signals package also contains an interface class for shared constants.

Constants

Overview

Constants is an interface containing state, status, mode, and option assignments shared by all blocks and their parts.

Attributes

Manual, auto, supervisory (*mas*) output modes are:

O_S Block is out of service.

INITIALIZE Transition actual mode enables bumpless transfer to target mode.

MANUAL Operator or set method can set output.

AUTO Block algorithm sets output.

SUPER Supervisory task sets output.

LINKED Permitted attribute indicating Mode target is connected to external source.

INTERLOCK Interlock sets output, actual mode is enabled by *trigger*, disabled by *reset*.

Local, remote, supervisory (*lrs*) setpoint modes are:

O_S Block is out of service

INITIALIZE Transition actual mode enables bumpless transfer to target mode

LOCAL Operator or set method can set setpoint.

REMOTE External block sets setpoint.

SUPER Supervisory task sets setpoint.

LINKED Permitted attribute indicates mode target connected to external source.

Signal quality *status* has the following (SEVA) ranked values:

CLEAR Signal is valid, on-line, and within spec.

CLOSE_CAS *status* is used to close cascade connection.

INIT *status* is used to open cascade connection.

BLURRED Signal is degraded, *uncertainty* is increased.

DAZZLED Signal is temporarily invalid, replaced by an estimate.

BLIND Signal is permanently invalid, replaced by an estimate.

FAILSAFE *status* indicates valve is forced to failsafe position.

OFFLINE Source is out of service.

Device *maintenance* status has the following (SEVA) ranked values:

NOMINAL Maintenance is not needed.

ON_TEST Device is in test or calibration mode.

LOW_PRIORITY Minor fault has been detected, quality *status* is BLURRED.

HIGH_PRIORITY Major fault has been detected, quality *status* is BLIND.

CRITICAL Critical fault detected, jeopardizing safety.

UNCHECKED *maintenance* reporting is not supported.

Limit indication and linking and setting permissions are contained in a packed Boolean status:

LIM_LO	Signal is at its low limit.
LIM_HI	Signal is at its high limit.
CONSTANT	Signal is manually set.
CASCADE	Signal is cascade connected.
UNSETABLE	Signal can not be set. It is linked and not MANUAL or LOCAL
LINKED	Signal is already linked, it cannot be connected to two sources.
LINK_NOSET	Signal can not be set or linked.
NON_NEG	Signal is prevented from having a negative value.
NOT_MEDIAN	Signal is determined not to be the median signal.

Input part options include:

SP_PV_TRK_MAN	Setpoint tracks PV when output is manually set.
SP_DN_UNCER	Setpoint is decreased by the *uncertainty*.
SP_UP_UNCER	Setpoint is increased by the *uncertainty*.
MULT_FF	Feedforward is multiplicative, else it is additive.
I_O	*in* is provided from *device* and *point*

Output part options include:

LIM_MAN	Limits are applied to MANUAL output.
ALM_MAN	Alarming is applied to a MANUAL output.
MAN_START	Block returns from out-of-service in MANUAL mode.
MULT_FF	Feedforward is multiplicative, else it is additive.
ALARM_BLURRED	Block alarm triggered by BLURRED signal *status*.
I_O	*out* is provided to *device* and *point*.

Feedback tuner states are;

PRETUNE	*Pretune* is active. Controller must be in MANUAL.
OFF	Tuner is off or not connected.
HOLD	*Selftune* is suspended but stored tuning sets are used.

QUIET	The loop is undisturbed, no error-peak search is in progress.
LOCATE_1	Error peak 1 is sought.
LOCATE_2	Error peak 2 is sought.
LOCATE_3	Error peak 3 is sought.
LOCATE_4	Error peak 4 is sought.
WAIT	Tuning update is waiting for an output peak search to complete.
SETTLE	*Selftune* is testing for error settling before returning to QUIET.

The feedforward tuner states are:

OFF	Tuner is off or not connected.
HOLD	Feedforward tuning is suspended and stored tuning sets are used.
QUIET	The loop is undisturbed, no moment integration is in progress.
UNMEAS	An error upset is detected before a load upset.
MEAS	A load upset is detected while QUIET.
SIGNIF	When MEAS, the control error exceeds the threshold
CNFIRM	When SIGNIF, the feedback tuner finds peak 1.
WAIT	There is no wait time before compensation update.
SETTLE	The error is tested for settling before returning to QUIET.

The alarmSum states are:

LL_ACT	Low-low alarm is active.
LO_ACT	Low alarm is active.
HI_ACT	High-high alarm is active.
HH_ACT	High alarm is active.
B_ACT	Boolean alarm is active.
BLK_ACT	Block alarm is active.
LL_UNACK	Low-low alarm is unacknowledged.
LO_UNACK	Low alarm is unacknowledged.

HI_UNACK High-high alarm is unacknowledged.
HH_UNACK High alarm is unacknowledged.
B_UNACK Boolean alarm is unacknowledged.
BLK_UNACK Block alarm is unacknowledged.
UNACK Set to false to acknowledge all of these alarms.

Mode

Overview

The cascade *input* parts use the Mode class for local-remote-supervisory (*lrs*) switching. The *output* parts use the Mode class for manual-auto-supervisory (*mas*) switching.

Attributes

The Mode class has four byte attributes whose bit assignments are assigned by the Constants interface:

target requested mode; one bit is set, default O_S bit set.
actual currently active mode; one bit is set, default O_S bit set.
permitted allowed modes; several bits may be set, default all bits are set.
last the last mode both requested and active; one bit is set, default O_S bit set.

Methods

getMode

Arguments: none.
Returns: active mode.

setMode

Arguments: desired mode.
If the mode is not linked
target mode is set to desired mode.
Returns: true
Else:
target mode is not changed.
Returns: false

linkTo Called from source

Arguments: sink reference.

If the sink is not linked

Source and sink are linked.

Returns: source reference

Else:

Returns: unchanged sink reference

unlink Called from source

Arguments: new sink reference.

Source and sink are unlinked

Returns: instantiates new sink object with existing target, actual, permitted (not linked), and last.

Parameter

Overview

Parameter is an abstract class providing features common to FloatParameter, BoolParameter, and IntParameter.

Attributes

status (SEVA ranked) signal quality status, enumerated byte.

limStatus Limit indication and link and set permissions, packed boolean byte.

source source is a String name.

time timestamp, a long integer msec. since 1970

Methods

getStatus

Arguments: none.

Returns: signal quality *status*.

FloatParameter

Overview

A block's or part's continuous tuning parameters are instances of the FloatParameter class.

It inherits from the Parameter class. The FloatParameter contains floating-point value and two status bytes. The value of an unlinked parameter may be set with the *setValue* method. A source for a linked parameter may be the *forw* or *back* part source

block's *out* variable. It is linked (connected) by its *linkTo* method with a reference for the sink block's parameter.

Attributes

value floating point signal value.

Attributes from super class

status (SEVA ranked) signal quality status, enumerated byte.

limStatus Limit indication and link and set permissions, packed boolean byte.

source source is a String name.

time timestamp, a long integer msec. since 1970

Methods

linkTo Called from source

Arguments: sink reference, source name.

If the sink is not linked

Source and sink are linked.

The *source* name is assigned

Returns: source reference

Else:

Returns: unchanged sink reference

unlink Called from source

Arguments: sink reference.

Source and sink are unlinked

Returns: instantiates new sink object with existing *value* and clear *status*.

getValue

Arguments: none.

Returns: value.

setValue

Arguments: desired value.

If the mode is not unsetable:

value is set to desired value.

Returns: true

Else:

value is not changed.

Returns: false

Methods from super class

getStatus

Arguments: none.

Returns: signal quality *status*.

FloatRange

Overview

The FloatRange class contains the high and low range limits and their units. A range object is contained in a FloatVariable. Range information is entered at the signal source and propagated downstream with a unidirectional connection and upstream with a bidirectional (cascade) connection.

Attributes

hi high range limit in engineering units, float, default is 100.

lo low range limit in engineering units, float, default 0.

units engineering units, String, default %.

delta is a communication threshold, a float value in engineering units, default 0.

Variable

Overview

Variable is an abstract class providing features common to Float Variable, Bool Variable, and Int Variable.

Attributes

alarmSum active and unacknowledged alarms, packed Boolean short.

maintenance SEVA ranked maintenance status, enumerated byte.

source source is a String name.

Methods

ack

Arguments: none.

All alarms are acknowledged.

Returns: none.

getAlarmHI

Arguments: none.

If high alarm is active:

Returns: true.

Else

Returns false.

getAlarmHH

Arguments: none.

If high-high alarm is active:

Returns: true.

Else

Returns false.

getAlarmLO

Arguments: none.

If low alarm is active:

Returns: true.

Else

Returns false.

getAlarmLL

Arguments: none.

If low-low alarm is active:

Returns: true.

Else

Returns false.

getAlarmBLK

Arguments: none.

If block alarm is active:

Returns: true.

Else

Returns false.

getAlarmB

Arguments: none.

If boolean alarm is active:

Returns: true.

Else

Returns false.

FloatVariable

Overview

A block output or input signal is an instance of the FloatVariable class. It inherits from Variable and is usually contained in the source block's *output* part and linked (connected) by its *linkTo* method with a reference for the sink block's variable. A cascade connection, such as primary-*out* to secondary-*remoteSP*, is bidirectional. A FoatParameter (*forw*) is set by the primary. Another FoatParameter (*back*) is set by the.secondary. Both parameters and other entities are contained in the FloatVariable class. A connection from an AnalogInput's *out* to an AdvancedPID's *in* is uni-directional. It's FloatVariable has a *forw* parameter but no *back* parameter. The FloatVariable also conveys *uncertainty*, *range*, and *maintenance* information, backwards for a cascade connection and forwards for a uni-directional connection. Alarm summary information is conveyed forwards.

Attributes

forw forward propagated FloatParameter, value (eu), status, limStatus.

back backward propagated FloatParameter, value (eu), status, limStatus.

range high, low range limits in engineering units (eu), FloatRange class.

uncertainty SEVA uncertainty, float value (eu).

Attributes from super class

alarmSum active and unacknowledged alarms, packed Boolean short.

maintenance SEVA ranked maintenance status, enumerated byte.

source source is a String name.

Methods

linkTo Called from *forw* source

Arguments: *forw* sink reference, source name.

If the *forw* sink is not linked

Source and sink are linked.

The *source* name is assigned.

Returns: *forw* source reference

Else:

Returns: unchanged *forw* sink reference

Unlink Called from *forw* source

Arguments: *forw* sink reference.

CASCADE or 0.

Source and sink are unlinked

Returns: instantiates new sink object.

Methods from super class

ack

Arguments: none.

All alarms are acknowledged.

Returns: none.

getAlarmHI

Arguments: none.

If high alarm is active:

Returns: true.

Else

Returns false.

getAlarmHH

Arguments: none.

If high-high alarm is active:

Returns: true.

Else

Returns false.

getAlarmLO

Arguments: none.

If low alarm is active:

Returns: true.

Else

Returns false.

getAlarmLL

Arguments: none.

If low-low alarm is active:

Returns: true.

Else

Returns false.

getAlarmBLK

Arguments: none.

If block alarm is active:

Returns: true.

Else

Returns false.

BoolParameter

Overview

A block's or part's binary parameters are instances of the BoolParameter class. It inherits from the Parameter class. The BoolParameter contains boolean value and two status bytes. The value of an unlinked parameter may be set with the *setValue* method. A source for a linked parameter may be the *forw* or *back* part source block's *out* variable. It is linked (connected) by its *linkTo* method with a reference for the sink block's parameter.

Attributes

value boolean signal value.

Attributes from super class

status (SEVA ranked) signal quality *status*, enumerated byte.

limStatus limit indication and link and set permissions, packed Boolean byte.

source source is a String name.

time timestamp, a long integer msec. since 1970

Methods

linkTo Called from source

Arguments: sink reference, source name.

If the sink is not linked.

Source and sink are linked.

The *source* name is assigned

Returns: source reference.

Else:

Returns: unchanged sink reference.

unlink Called from source

Arguments: sink reference.

Source and sink are unlinked

Returns: instantiates new sink object with existing *value* and clear *status*.

getValue

Arguments: none.

Returns: *value*.

setValue

Arguments: desired value.

If the parameter is not unsetable:

value is set to desired value.

Returns: true

Else:

value is not changed.

Returns: false

Methods from super class

getStatus

Arguments: none.

Returns: signal quality *status*.

BooleanVariable

Overview

A block's binary output or input signal is an instance of the BoolVariable class. It inherits from Variable and is usually contained in the source block's *output* part and linked (connected) by its *linkTo* method with a reference for the sink block's variable. A cascade connection, such as primary-*out* to secondary-*remoteSP*, is bidirectional. A BoolParameter (*forw*) is set by the primary. Another BoolParameter (*back*) is set by the secondary. Both parameters and other entities are contained in the BoolVariable class. The BoolVariable also conveys the *maintenance* information, backwards for a cascade connection and forwards for a uni-directional connection. Alarm summary information is conveyed forwards.

Attributes

forw forward propagated BoolParameter, *value* (eu), *status*, *limStatus*.

back backward propagated BoolParameter, *value* (eu), *status*, *limStatus*.

Attributes from super class

alarmSum active and unacknowledged alarms, packed boolean short.

maintenance SEVA ranked *maintenance* status, enumerated byte.

source source is a String name.

Methods

linkTo Called from *forw* source

Arguments: *forw* sink reference, source name.

If the *forw* sink is not linked

Source and sink are linked.

The *source* name is assigned.

Returns: *forw* source reference

Else:

Returns: unchanged *forw* sink reference

unlink Called from *forw* source

Arguments: *forw* sink reference

CASCADE or 0

Source and sink are unlinked

Returns: instantiates new sink object.

Methods from super class

ack

Arguments: none.

The boolean alarm is acknowledged.

Returns: none.

getAlarm

Arguments: none.

If boolean alarm is active:

Returns: true.

Else

Returns false.

getAlarmBLK

Arguments: none.

If block alarm is active:

Returns: true.

Else

Returns false.

IntParameter

Overview

A block's or part's integer parameters are instances of the IntParameter class. It inherits from the Parameter class. The IntParameter contains integer value and two status bytes. The value of an unlinked parameter may be set with the setValue method. A source for a linked parameter may be the *forw* or *back* part source block's *out* variable. It is linked (connected) by its linkTo method with a reference for the sink block's *parameter*.

Attributes

value integer signal value.

Attributes from super class

status (SEVA ranked) signal quality *status*, enumerated byte.

limStatus limit indication and link and set permissions, packed Boolean byte.

source source is a String name.

time timestamp, a long integer msec. since 1970

Methods

linkTo Called from source

Arguments: sink reference, source name.

If the sink is not linked

Source and sink are linked.

The source name is assigned.

Returns: source reference.

Else:

Returns: unchanged sink reference.

unlink Called from source

Arguments: sink reference.

Source and sink are unlinked

Returns: new sink reference with existing value and clear status.

getValue

Arguments: none.

Returns: *value.*

setValue

Arguments: desired value.

If the parameter is not unsetable:

value is set to desired value.

Returns: true

Else:

value is not changed.

Returns: false

Methods from super class

getStatus

Arguments: none.

Returns: signal quality *status*.

IntRange

Overview

The IntRange class contains the high and low range limits. A range object is contained in an IntVariable. Range information is entered at the signal source and propagated downstream with a unidirectional connection and upstream with a bidirectional (cascade) connection.

Attributes

hi high range limit in integer, default is 1.

lo low range limit in integer, default is 0.

IntVariable

Overview

A block's integer output or input signal is an instance of the IntVariable class. It inherits from Variable and is usually contained in the source block's *output* part and linked (connected) by its linkTo method with a reference for the sink block's variable. A cascade connection, such as primary-*out* to secondary-*remoteSP*, is bidirectional. An IntParameter (*forw*) is set by the primary. Another IntParameter (*back*) is set by the secondary. Both parameters and other entities are contained in the IntVariable class. The IntVariable also conveys range, and maintenance information, backwards for a cascade

connection and forwards for a uni-directional connection. Alarm summary information is conveyed forwards.

Attributes

forw forward propagated BoolParameter, value (eu), status, limStatus.

back backward propagated BoolParameter, value (eu), status, limStatus.

range high, low range limits in engineering units (eu), IntRange class.

Attributes from super class.

alarmSum active and unacknowledged alarms, packed Boolean short.

maintenance SEVA ranked maintenance status, enumerated byte.

source source is a String name.

Methods

linkTo Called from *forw* source

Arguments: *forw* sink reference, source name.

If the *forw* sink is not linked

Source and sink are linked.

The *source* name is assigned.

Returns: *forw* source reference

Else:

Returns: unchanged *forw* sink reference

unlink Called from *forw* source

Arguments: *forw* sink reference, CASCADE or 0

Source and sink are unlinked

Returns: instantiates new sink object.

Methods from super class

ack

Arguments: none.

All alarms are acknowledged.

Returns: none.

getAlarmHI

Arguments: none.

If high alarm is active:

Returns: true.

Else

Returns false.

getAlarmHH

Arguments: none.

If high-high alarm is active:

Returns: true.

Else

Returns false.

getAlarmLO

Arguments: none.

If low alarm is active:

Returns: true.

Else

Returns false.

getAlarmLL

Arguments: none.

If low-low alarm is active:

Returns: true.

Else

Returns false.

getAlarmBLK

Arguments: none.

If block alarm is active:

Returns: true.

Else

Returns false.

Discussion of Signals Classes

Interblock connections are performed with Variable and Parameter objects contained in the upstream block and linked by reference to the downstream block. When the upstream block and downstream block are in different stations, it is necessary to have a periodically updated proxy Variable or Parameter in the station with the downstream block. Fixed constants shared by potentially all blocks and parts are made available in the Constants interface implemented or inherited by all PCOs. All PCOs also inherit or

implement the Serializable interface. Composites and blocks also inherit or implement the Runnable interface.

External Interfaces for Signals Classes

Output signals are objects of a float, boolean, or int Variable class contained in the upstream block. An input Variable of one block may be linked by reference to an output Variable contained in another block in the same station. When the output variable is contained in a block from a different station it is necessary to store a periodically updated copy of the output Variable in the same station as the downstream block so that the input may be linked by reference to the copy (proxy). A bidirectional Variable involved in a cascade connection requires that certain of its Parameters and attributes (forw and alarmSum) be communicated in the downstream direction, while others (back and range) are communicated in the upstream direction.

Parameters of one block may be linked by reference to a Parameter component of an output Variable in another block. Similarly an updated proxy Parameter is required in the downstream block's station if the upstream block is in a different station.

A PCO Temperature Cascade Control Loop with a Simulated Process

A further appreciation of the operation of a control system 10 (Figure 1) and particularly, for example, of a controller 36 (or other control device) that utilizes process control objects (PCOs) of the type described above may be attained by reference to the Java code listing that follows. This is an application of a PCO that implements a temperature cascade control loop on the device (e.g., controller 36) on which it resides. A graphical depiction of the operation of that control loop is presented in Figure 16, above.

Referring to the listing, in lines 3 - 7 declare and instantiate the composites and blocks that make up the temperature cascade control loop PCO, to wit, supervisor (Supv) block, the cascase composite (Cascade), the temperature process composite (TempProcess), the information collection block (InfoCollect), and the object (ObjList) that enumerates all objects that are to be executed by the control system 10. The latter object is referred to as the "run" list. In lines 9 - 10, the code declares alias names for parameters internal to the TempCasc PCO object itself.

Line 13 begins the configuration method, which is provided here merely as an example of how a previously-instantiated PCO (e.g., the TempCasc PCO) can be customized for use in a particular control system. Those skilled in the art will, of course, appreciate that configuration can be performed other than via "hard coding" as shown in lines 13, *et seq*. Typically, such configuration is performed via a graphical user interface.

In line 16, the previously instantiated user block object (s21) is added to the run list. No additional configuration is shown here with respect thereto.

In lines 19 - 33, the cascade composite object (tc21) is configured and its input variables connected, by reference, to the outputs of other objects utilized in the system 10. Particularly, in lines 19 - 20, the analog input of the primary control loop in temperature cascade object tc21 is connected to the temperature measurement output of the simulated process object (pr21). In lines 21 - 22, a bidirectional connection is established between the PID supervisory setpoint of the composite object tc21 and the output of the supervisory block s21. In lines 23 - 24, a unidirectional connection is established between the flow measurement output of the simulated process object pr21 and the analog input of the secondary control loop in the temperature cascade object tc21.

In step 25, the configuration method of the temperature cascade control object tc21 is invoked. This effects initialization of the parameters and connections for that object tc21.

In step 26 - 27, an optional alarm part of object tc21's primary loop analog input is instantiated. The default alarm limits are set in accord with the output range of the analog input. In steps 28 - 29, the high and low alarm limits are set to 75 and 30, respectively.

In step 30, the input mode of the object tc21's primary PID is set to "supervisory," indicating that it will take its setpoint from supervisory object s21. Those skilled in the art will, of course, appreciate that the supervisory setpoint could be set by other external sources.

In steps 31 - 32, the tuning settings previously defined in user block s21 are inserted into the primary and secondary loops of the PID controllers of the composite tc21. In step 33, the block period of the analog input block of the secondary loop is set.

In step 34, the composite tc21 is added to the run list.

In steps 37 - 40, the secondary analog output of the simulated process block tc21 is connected to the flow process input of composite pr21, the latter's configure method is called and it is added to the run list.

In lines 43 - 47, inputs for the information collection block are connected to outputs of blocks and composites of interest, and the former is added to the run list.

In lines 49 - 50, the composite objects tc21 and pr21 are turned "on" for execution.

The run method of the temperature cascade control loop begins at line 54. This method is called by the control system 10 scheduler each BPC based on the loop's inclusion in the run list. In steps 64 - 78, the run method of the cascade control loop invokes the run methods of its respective composites and blocks, specifically, those enumerated the run list. Though not shown here, the composites have their own run lists, the blocks on which are called as each composite is called. In the illustrated embodiment, a loop beginning at step 62 limits the foregoing to *n* cycles, merely for purposes of testing and illustration.

```
public class TempCasc implements Constants
{
  static Supv           s21  = new Supv();
  static Cascade        tc21 = new Cascade();
  static TempProcess    pr21 = new TempProcess();
  static InfoCollect    ic21 = new InfoCollect(10,0,4,0,0,0);
  static ObjList        comp = new ObjList(10);

  static FloatParameter    tc21_temp;
  static FloatParameter    tc21_flow;


  static void config()
  {
    // User block
    comp.add(s21);

    // Cascade Object
    tc21.getPri().getAin().getInput(0).in = pr21.temp.getOutput().getOut().linkTo(
       tc21.getPri().getAin().getInput(0).in);
    tc21.getPri().getPid().getInput().supvSP = s21.getOutputFloat(0).getOut().linkTo(
       tc21.getPri().getPid().getInput().supvSP);
    tc21.getSec().getAin().getInput(0).in = pr21.flow.getOutput().getOut().linkTo(
       tc21.getSec().getAin().getInput(0).in);
    tc21.config();
    tc21.getPri().getAin().getOutput().alarmHi = new Alarm(
       tc21.getPri().getAin().getOutput().getOut().range);
    tc21.getPri().getAin().getOutput().alarmHi.getHi().setValue(75.);
    tc21.getPri().getAin().getOutput().alarmHi.getLo().setValue(30.);
    tc21.getPri().getPid().getInput().getLrs().setMode(SUPER);
    tc21.getPri().getPid().cntl = s21.parTC;
```

```
tc21.getSec().getPid().cntl = s21.parFC;
tc21.getSec().getAin().setBlockPeriod(.1);
comp.add(tc21);

// TempProcess
pr21.flow.getInput(0).in = tc21.getSec().getAout().getOutput().getOut().linkTo
    (pr21.flow.getInput(0).in,~CASCADE);
pr21.config();
comp.add(pr21);

// InfoCollect
ic21.paramFloat[0] = tc21.getSp().linkTo(ic21.paramFloat[0]);
ic21.paramFloat[1] = tc21.getPv1().linkTo(ic21.paramFloat[1]);
ic21.paramFloat[2] = tc21.getPv2().linkTo(ic21.paramFloat[2]);
ic21.paramFloat[3] = tc21.getOut().linkTo(ic21.paramFloat[3]);
comp.add(ic21);

tc21.setOffOn(true);
pr21.setOffOn(true);

} // end of config()

public void run()
{
  int n,i,ii;
  double x;
  long timeStart, dif;
  n = 30;

  timeStart = System.currentTimeMillis();
  for (i = 0; i<=n; i++)
```

```
63      {
64          for (int j = 0; j<fc22.getBpcPerPeriod(); j++)
65          {
66              for (int k = 0; k<comp.getLength(); k++)
67              {
68                  if (comp.getList(k)==null) break;
69                  comp.getList(k).run();
70              }
71              timeStart += 100L;
72              dif = timeStart-System.currentTimeMillis();
73              if (dif>0)
74              {
75                  try {Thread.sleep(dif);}
76                  catch(InterruptedException e) {System.out.println(e);}
77              }
78          }
79      } // end of run()
80
81      public static void main(String args[])
82      {
83          TempCasc device = new TempCasc();
84          Thread     thd1 = new Thread(device);
85
86          device.config();
87          thd1.start();
88      }
89      }
90
```

Summary

Described herein are control systems and method meeting the objects set forth above. Those skilled in the art will appreciate that the illustrated embodiment is an example of the invention and that other embodiments incorporating changes therein fall within the scope of the invention. Thus, by way of example, it will be appreciated that the invention can be practiced utilizing, instead of or in addition to Java objects, Java applets, servlets, and/or software constructs of programming languages other than Java. Moreover, it will be appreciated that the invention can be utilized in a range of control application, including, process, industrial, and environmental, among others. In view thereof, what we claim is:

Appendix

Appendix

to

Patent Application for

Methods and Apparatus for Object-Based Process Control

Copies of common specifications for

United States Patent Application Serial No. 09/591,604, and to counterpart PCT Application Serial No. PCT/US 00/15860, both filed June 9, 2000, entitled METHODS AND APPARATUS FOR CONTROL USING CONTROL DEVICES THAT PROVIDE A VIRTUAL MACHINE ENVIRONMENT AND THAT COMMUNICATE VIA AN IP NETWORK, and

United States Patent Application No. 09/345,215, filed June 30, 1999, and its counterpart PCT Application Serial No. US00/16152, filed June 9, 2000, both entitled Process Control and Method with Auto-Updating

Appendix

METHODS AND APPARATUS FOR CONTROL USING CONTROL DEVICES THAT PROVIDE A VIRTUAL MACHINE ENVIRONMENT AND THAT COMMUNICATE VIA AN IP NETWORK

Background of the Invention

This application claims the priority of the following United States Patent Applications: United States Patent Application Serial No. 60/139,071, entitled OMNIBUS AND WEB CONTROL, filed June 11, 1999; United States Patent Application Serial No. 60/144,693, entitled OMNIBUS AND WEB CONTROL, filed July 20, 1999; United States Patent Application Serial No. 60/149,276, entitled METHODS AND APPARATUS FOR PROCESS CONTROL ("AUTOARCHITECTURE"), filed August 17, 1999; United States Patent Application Serial No. 09/345,215, entitled PROCESS CONTROL SYSTEM AND METHOD WITH IMPROVED DISTRIBUTION, INSTALLATION, AND VALIDATION OF COMPONENTS, filed June 30, 1999.

The invention pertains to control systems and, more particularly, to methods and apparatus for networking, configuring and operating field devices, controllers, consoles and other control devices.

The terms "control" and "control systems" refer to the control of a device or system by monitoring one or more of its characteristics. This is used to insure that output, processing, quality and/or efficiency remain within desired parameters over the course of time. In many control systems, digital data processing or other automated apparatus monitor a device, process or system and automatically adjust its operational parameters. In other control systems, such apparatus monitor the device, process or system and display alarms or other indicia of its characteristics, leaving responsibility for adjustment to the operator.

Control is used in a number of fields. Process control, for example, is employed in the manufacturing sector for process, repetitive and discrete manufactures, though, it also has wide application in utility and other service industries. Environmental control finds application in

Appendix

residential, commercial, institutional and industrial settings, where temperature and other environmental factors must be properly maintained. Control is also used in articles of manufacture, from toasters to aircraft, to monitor and control device operation.

Modern day control systems typically include a combination of field devices, controllers, workstations and other more powerful digital data processing apparatus, the functions of which may overlap or be combined. Field devices include temperature, flow and other sensors that measure characteristics of the subject device, process or system. They also include valves and other actuators that mechanically, electrically, magnetically, or otherwise effect the desired control.

Controllers generate settings for the control devices based on measurements from sensor type field devices. Controller operation is typically based on a "control algorithm" that maintains a controlled system at a desired level, or drives it to that level, by minimizing differences between the values measured by the sensors and, for example, a setpoint defined by the operator.

Workstations, control stations and the like are typically used to configure and monitor the process as a whole. They are often also used to execute higher-levels of process control, e.g., coordinating groups of controllers and responding to alarm conditions occurring within them.

In a food processing plant, for example, a workstation coordinates controllers that actuate conveyors, valves, and the like, to transport soup stock and other ingredients to a processing vessel. The workstation also configures and monitors the controllers that maintain the contents of that vessel at a simmer or low boil. The latter operate, for example, by comparing measurements of vapor pressure in the processing vessel with a desired setpoint. If the vessel pressure is too low, the control algorithm may call for incrementally opening the heating gas valves, thereby, driving the pressure and boiling activity upwards. As the pressure approaches the desired setpoint, the algorithm requires incrementally leveling the valves to maintain the roil of the boil.

Appendix

The field devices, controllers, workstations and other components that make up a process control system typically communicate over heterogeneous media. Field devices connect with controllers, for example, over dedicated "fieldbuses" operating under proprietary or industry-specific protocols. Examples of these are FoxCom(TM), Profibus, ControlNet, ModBus, DeviceNet, among others. The controllers themselves may be connected to one another, as well as to workstations, via backplane or other proprietary high-speed dedicated buses, such as Nodebus(TM). Communications among workstations and plant or enterprise-level processors may be via Ethernet networks or other Internet Protocol (IP) networks.

Control device manufacturers, individually, and the control industry, as a whole, have pushed for some uniformity among otherwise competing communication standards. The Foundation Fieldbus, for example, is the result of an industry-wide effort to define a uniform protocol for communications among processor-equipped (or "intelligent") field devices. Efforts such as this have been limited to specific segments of the control hierarchy (e.g., bus communications among field devices) and are typically hampered by technological changes that all to soon render the standards obsolete.

Still less uniform are the command and operation of control devices. Though field devices may function at the direction of controllers and controllers, in turn, at the direction of workstations (or other plant-level processors), proprietary mechanisms within the individual components determine how they perform their respective functions. Even the commands for invoking those functions may be manufacturer- or product-specific. Thus, the commands necessary to drive actuators of one manufacturer will differ from those of another. How the corresponding commands are processed internally within the actuators differ still more (though, hopefully, the results achieved are the same). The specific programming codes used to effect a given control algorithm likewise differs among competing makes, as do those of the higher-level control processors.

Industry efforts toward harmonization of software for command and operation of control devices have focused on editing languages that define process control algorithms. This is distinct from the codes used to effect those algorithms within control devices and, rather, concerns software

Appendix

"tools" available to users to specify the algorithms, e.g., editors including IEC-1131 standard languages such as Field Blocks, Sequential Function Charts (SFC), Ladder Logic and Structured Text.

Less concerted are industry moves to extend monitoring and limited configuration capabilities beyond in-plant consoles, e.g., to remote workstations. An example of this was the abortive Java for Distributed Control (JDC) effort, which proposed enabling in-plant workstations to serve web pages to remote Java bytecode-enabled client computers. The latter used the to web pages to monitor and set control parameters which the workstations, in turn, incorporated into their own control schemes.

An academic system along these same lines was suggested by the Mercury Project of the University of Southern California, proposing the use of a web browser to enable remote users to control a robotic arm via a server that controlled the arm. A related company-specific effort included that announced by Tribe Computer Works that allegedly enabled users to manage routers and remote access servers over IP networks using web browser software. See, "Tribe Defines Net Management Role For Web Browser Software," Network World, May 22, 1995, at p. 14.

Thus sets the stage for the present invention, an object of which is to provide improved methods and apparatus for networking, configuring and operating field devices, controllers, consoles and other control devices. A related object is to provide such methods and apparatus for process control.

Further objects of the invention are to provide such methods and apparatus as reduce the confusion, complexity and costs attendant to prior art control systems.

Related objects of the invention are to provide such methods and apparatus as can be implemented with commercial off the shelf hardware and software.

Appendix

Still further objects of the invention are to provide such methods and apparatus as achieve confusion-, complexity- and cost-reduction without hampering manufacturer creativity and without removing incentives to development of product differentiators.

Summary of the Invention

The foregoing are among the objects attained by invention which provides, in one aspect, an improved field device for a process or other control system. The field device includes a virtual machine environment for executing Java byte code (or other such intermediate code) that, for example, configures the device to execute a control algorithm.

By way of non-limiting example, the field device can be an "intelligent" transmitter or actuator that includes a low power processor, along with a random access memory, a read-only memory, FlashRAM, and a sensor interface. The processor can execute a real-time operating system, as well as a Java virtual machine (JVM). Java byte code executes in the JVM to configure the field device to perform typical process control functions, e.g., for proportional integral derivative (PID) control and signal conditioning.

Further aspects of the invention provide a field device, such as a low-power intelligent actuator, that incorporates an embedded web server. This can be used to configure, monitor and/or maintain the device itself (as well as other elements of the control system) via a browser attached directly to the device or coupled to it over the network. To this end, the field device can incorporate a configuration editor, e.g., operating on a processor within the field device, that an end-user executes via the browser and web server.

Such a configuration editor can, in related aspects of the invention, be enabled or disabled depending on the environment in which it is used and more specifically, for example, the type of network in which it is incorporated. Thus, for example, the editor can be disabled when the field device is incorporated in a process control network that includes, e.g., an applications development environment suitable for configuration of the field device. Conversely, it can be enabled when the field device is incorporated in a network that lacks such a capability.

Appendix

Still further aspects of the invention provide a field device as described above that includes an interface to an IP network, through which the device communicates with other elements of the control system. The IP network can be, for example, an Ethernet network. Moreover, it can be "powered," carrying electrical power as well as packets, datagrams, and other control or data signals. The field device, in related aspects of the invention, draws operational power, e.g., for its processor and other components, from such a network.

Yet further aspects of the invention provide a field device as described above that obtains configuration information and/or its network address from such an IP network upon start-up. To this end, on power-up or coupling to the network, the field device can supply an identifier (e.g., attained from a letterbug, assigned by a hub, or otherwise) to a DHCP or other server on the network. Once provided with an IP address, the field device can formally enter into the control network, e.g., by posting its characteristics to a network bulletin board, e.g., using a network enabler such as a Jini and/or JavaSpace server, or the like. Other network devices monitoring or notified via such a bulletin board can send configuration information to the field device or otherwise.

Still further aspects of the invention provide control devices, such as servers, control stations, operator consoles, personal computers, handheld computers, and the like, having attributes as described above. Such control devices can have other attributes, according to further aspects of the invention. Thus, by way of non-limiting example, they can provide web servers that collect process data from one or more control devices, generate source for operator displays, provide access to the control system, and host an applications development environment.

Still other aspects of the invention provide process, environmental, industrial and other control systems that comprise field and control devices as described above that are coupled via an IP network and, particularly, for example, by a powered IP network.

Additional aspects of the invention are directed to DHCP servers and network enablers (optionally, including web servers) for use in control systems as described above. Related

Appendix

aspects provide such servers and enablers as are embodied in solid state technologies, e.g., with no moving parts.

These and other aspects of the invention are evident in the attached drawings, and in the description and claims that follow.

Brief Description of the Drawings

A more complete understanding of the invention may be attained by reference to the drawings, in which:

Figure 1 depicts a process control system 10 according to one practice of the invention;

Figures 2 and 3 depict more particular embodiments of a control system of the type shown in Figure 1;

Figure 4 depicts a native intelligent field device according to the invention;

Figure 5 depicts a native control device according to the invention; and

Figure 6 depicts a native intelligent positioner implementation according to the invention.

Detailed Description of the Illustrated Embodiment

Figure 1 depicts a process control system 10 according to the invention. The system includes networked control devices that monitor and control a hypothetical mixing process that utilizes mixing chamber 22, fluid inlets 24, 26, fluid outlet 28, paddle 30, cooler 32, and cooler inlet 34. Though illustrated and described below for use in connection with process control, those skilled in the art will appreciate that apparatus and methods according to the invention can be used in connection any industrial, manufacturing, service, environmental or other process, device or system amenable to monitoring or control (hereinafter, collectively, "control").

Appendix

The networked control devices include actuators, such as the valves depicted as controlling inlets and outlets 24 - 28 and 34. A further actuator is shown controlling paddle 30. These and other actuators utilized by the control system are constructed and operated in the conventional manner, as modified in accord with the teachings hereof. The actuators operate under control of respective field device controllers, labeled CTL, that are also constructed and operated in the conventional manner to provide initialization, signal conditioning and communications functions.

Rather than using separate controllers CTL, the actuators can be of the intelligent variety and can include integral microprocessors or other digital data processing apparatus for control, initialization, signal conditioning, communications and other control-related functions. For sake of convenience, the label CTL is used regardless of whether the control-related functionality is integral to the actuators (e.g., as in the case of intelligent actuators) or otherwise.

Illustrated sensor 29 monitors a temperature, level or other characteristic of fluid in chamber 22. The sensor 29, as well as other sensing apparatus utilized by the system, are constructed and operated in the conventional manner known in the art, as modified in accord with the teachings hereof. They can be coupled to the control network via a transmitter or other interface device INT that, too, is constructed and operated in the conventional manner, as modified by the teachings hereof. The interface devices facilitate initialization, signal conditioning and communications between the sensors and the control system. As above, one or more sensors can be of the intelligent variety, incorporating integral microprocessors or other digital data processing capabilities for initialization, signal conditioning, communications and other control-related functions. Here, too, the label INT is used in reference to the control-related functionality, regardless of whether embodied in an intelligent transmitter or otherwise.

The networked control devices include one or more controllers 36 that monitor and control respective aspects of the hypothetical mixing process in the conventional manner, as modified in accord with the teachings hereof. The controllers can comprise mainframe computers, workstations, personal computers, special-purpose hardware or other digital data processing

Appendix

apparatus capable of performing conventional monitoring and control functions. Preferred controllers are constructed and operated in the manner of the CP control processors commercially available from the assignee hereof, as modified in accord with the teachings herein.

The control system 10 includes a variety of devices that serve as user interfaces and that provide configuration and/or control functions, all in the conventional manner as modified in accord with the teachings hereof. Illustrated for these purposes are workstation 40, laptop computer 42 and handheld computer 44. These devices can provide configuration and control functions directly, as in the case of workstation 40, or in cooperation with server devices, e.g., as in the case of handheld computer 44 and server 46. Apparatus 40 - 44 can couple with the control network directly, e.g., via bus or network connection, or indirectly, e.g., via satellite, wireless connection or modem connection.

The control devices 36 - 46, CTL and INT, collectively, referred to as "native" devices, are coupled for communications via a medium that permits at least selected ones of the devices to communicate with one another. To this end, in the illustrated embodiment those devices are coupled via one or more networks 48 that are, preferably, IP-based such as, by way non-limiting example, Ethernets. The network(s) can include, as indicated by the multiple segments shown in the drawing, multiple segments such as various wide and local area networks. They may also include high and/or low bandwidth components, such as phone lines, and low and/or high latency components, such as geosynchronous satellites networks.

Figure 2 depicts a more particular embodiment of a control system 50 of the type shown in Figure 1. The system includes an enterprise server 52, a first thin client 54, plant server 56, a second thin client 58, a controller 60, a Java-enabled field device 62, and one or more field devices 64, coupled to one another, e.g., in the manner illustrated, by one or more networks 66, 68.

Native enterprise server 52 (corresponding, by way of non-limiting example, to server 46) comprises a mainframe computer or engineering workstation that executes enterprise-level

Appendix

applications such as, by non-limiting example, those for financial, asset planning and procurement, distribution and/or human resources. In addition, it supports web serving, as well as optional object, relational and other database management systems. Server 52 executes Windows NT, Solaris or another conventional commercial or proprietary operating system. It is also equipped with a Java Virtual Machine (JVM), e.g., capable of executing Java virtual machine instructions (bytecodes), of performing remote method invocation (RMI), and of supporting Jini networking. The enterprise server 12 can be coupled to further networks, e.g., to the Internet, as shown, in any manner known in the art.

Native thin client 54 (corresponding, for example, to handheld computer 44) provides similar functionality as server 52, though its actual processing activity is limited to user input and output. Application processing, such as financial, asset planning and procurement, distribution and/or human resources, are performed on behalf of client 54 by a server, such as server 52. The operating system and JVM functions may be embedded in the conventional manner of a thin client. The thin client 54 is coupled to the enterprise server 52 over a business information network 66 (corresponding, for example, to network 48), typically, an Ethernet or other IP network, configured in a LAN, WAN or the like.

Native plant server 56 (corresponding, by way of non-limiting example to workstation 40) comprises a plant control console or engineering workstation modified in accord with the teachings hereof, executing plant-level control applications including system, process, engineering, plant information, configuration, lab quality, maintenance, resource and documentation applications of the type known in the art . Like enterprise server 52, plant server 56 can execute Windows NT, Solaris or other conventional operating systems. It is also preferably equipped to execute a Java Virtual Machine as described above. Plant server 56 is coupled to the enterprise server 52 and thin client 54 over the business information network 66.

Native thin client 58 provides similar functionality as server 56 though, again, relies on that (or another) server to perform most processing activity. As above, the operating system and JVM functions may be embedded in the conventional manner of a thin client. The thin client 58 is

Appendix

coupled to the plant server 56 over a control network 68 (corresponding, for example, to network 48), e.g., an Ethernet.

Native controller 60 (corresponding, for example, to controller 36) executes control algorithms to control associated non-native field devices 64, e.g., via any variety of commercial and/or proprietary field bus 70 hardware and protocols. Where processing resources are limited, the controller 60 utilizes a embedded operating system that supports web serving and the JVM. The controller 60 is coupled to the plant sever 66 and to the thin client 58 via control network 68.

Native field device 62 is a sensor, actuator or other field device. The illustrated device is of the intelligent variety, including processor (not shown) and operating system. It can be of the type commercially available in the marketplace, as modified in accord with the teachings hereof. The illustrated device supports web serving and JVM, as described above. The device 62 provides information collection and control functions, as illustrated.

Figure 3 depicts another a more particular embodiment of a control system 50 of the type shown in Figure 1.

Referring to Figure 1, the illustrated control system 10, 50 uses web browsers, Java, Jini, JavaSpaces, TCP/IP and other technologies normally associated with the Internet and the world wide web to create a self-defining control network that minimizes complexity while emphasizing the portability and re-usability of the user's application. These technologies are advantageously employed to eliminate proprietary hardware and software to preserve the user's investments; eliminate network configuration and system management through self-configuring networks; increase the user's choice of algorithm suppliers by implementing control in Java; preserve the user's applications through hardware and system software changes through the use of Java and Web Browsers for process displays; minimize the wiring costs; reduce maintenance by making intelligent field devices a practical reality.

In addition to web technologies, the illustrated control system 10 uses an object location service, a commercial messaging service, and standard networking equipment, such as hubs, routers,

Appendix

network interface cards, where applicable. It also relies on common standards, where applicable, such as 802.3, Internet RFCs, the IEEE 1451 sensor standards. The system 10 supports a heterogeneous computing environment and utilizes industry standards (e.g., Microsoft standards) to provide communications between the native control components to the business and desktop environments.

1 Device Hardware

1.1 Platform-Defining Control Devices

In the illustrated embodiment, native control devices such as controllers 36, 60, workstation 40, servers 46, 52, 56 providing a platform for the control system typically include a central processing unit (CPU), memory (RAM), network support hardware, access to permanent storage, an operating system, and a Java Virtual Machine (JVM) including the TCP/IP suite. In addition, the devices include web server software, e.g., software of the type that serves graphical "web" pages in response to requests by other devices. Configurator software can be provided, as well, permitting each device to configure the control system or selected portions of it. Those skilled in the art will appreciate that not all of these components need be included in all native control devices, e.g., some commercially available JVMs can serve as an OS themselves.

Process control object (PCO) software provided on the platform-defining control devices comprise a collection of data and methods implemented in Java and executed on the native control devices' JVMs to perform typical process control functions, by way of non-limiting example, signal conditioning and PID control. Likewise, station management object (SMO) software on the devices comprise data and methods that allow the device to report its health, performance, and other status information in a uniform manner. The SMO software can implement SNMP or the equivalent Java functionality.

Software is also provided on the platform-defining controls devices for messaging services for data transfer and alarm/event notification, as well as software comprising system management pages. Each control device may include additional software, of course, depending on its functionality.

Appendix

1.2 Field Devices

Native intelligent field devices typically include a low power CPU, e.g., a NetARM or Java Chip; a real-time OS like VxWorks, QNX, PSOS; RAM; FlashRAM to serve as permanent storage; ROM/EEPROM to serve as the home for the OS and other permanent software; an Ethernet interface; power from the Ethernet interface (in the event a powered Ethernet network or hub is used) or otherwise; a sensor interface , e.g., IEEE 1451.1 and 1451.2; JVM; a web server, and a device specific configurator servlet. A field device so constructed can be configured and monitored via a lightweight web browser, e.g., handheld computer 44, coupled to the device over the network 48.

The field devices can include serial interfaces to allow the attachment of these devices to HART, FoxComm, Profibus and other networks operating under a protocol different from that of network 48. Combined with appropriate software, these devices provide the user with a single transmitter suitable for use on any field network.

One configuration of a native intelligent field device including the elements described above these elements is shown in Figure 5. Those skilled in the art will appreciate that other configurations can be realized, as well, in accord with the teachings hereof.

1.2.1 Stoic Sensors

The control system 10 can include one or more stoic sensors, not illustrated, that constitute minimal sensing elements. Typically, these are simply silicon chips that are packaged to allow them to sense the process environment and that have only enough electronics to relay measurements to a pair of wires that carry the raw signal to another device for processing. Some stoic sensors generate a signal without external power; others require some excitation. In a preferred embodiment, native devices 36 - 46, CTL and INT according to the invention support the attachment of many stoic sensors.

1.2.2 I/O

1.2.2.1 Overview

Appendix

Some I/O devices, e.g., thermocouples, RTDs and analog transmitters, may not use a transmitter compatible with the network 48 but, rather, send raw sensor output to a multiplexor for processing. To accommodate these devices, the system 10 includes two types of intelligent I/O cards: network I/O cards supporting IP and an API, and native I/O cards that support the protocol of network 48 (i.e., the "native" protocol) directly.

Network I/O cards are coupled directly to network 48. Native control devices are IP enabled and support the proprietary API of the cards, thereby, permitting reading and writing of their I/O registers. The cards' respective APIs support retrieval of data in several formats (raw counts, linearized counts, engineering units, etc.), as well as the assignment of simple configuration information to those registers, if the device supports such functions. To this end, the native devices utilize I/O classes that corresponding to the native I/O protocols. Redundant I/O devices can be physically interfaced transparently through a single I/O card or through multiple independent network I/O cards. Logically, PCOs support at least the use of multiple independent cards. Alternative I/O features may be used in addition. Native I/O cards are substantially similar to native-enabled transmitters, except for packaging and scale. These I/O cards may be considered as small multi-loop controllers or multiplexors.

1.2.2.2 Foreign Device Integration

Networked I/O cards are utilized with devices that cannot otherwise directly interface with the network 48. The cards, in this case, provide an interface that permits at least reading and writing of relevant device values, which can be stored internally to the card and accessed via its API. Foreign devices that are control systems in its own right typically maintain objects or other data structures representing process values and associated data. A preferred interface to these "devices" are through objects that wrap the foreign function blocks with the services expected of native devices, , e.g., detail displays and configurators, which are accessed in exactly the same manner as native ones of those same services. For example, a name service is be able to locate foreign blocks and the native API of system 10 permits querying those blocks for values.

1.3 Native Controllers

Appendix

In a preferred embodiment, all native devices may be used as controllers, though, dedicated I/O-less native controllers, such as controllers 36 and 60, are typically required for unit-wide operations. Thus, for example, control can also be provided by personal computers, workstations and servers of the type shown as elements 40, 42, 46, in Figure 1. Native controllers, e.g., 36, 60 are of the same general design as native boards used in the transmitters and actuators. However, they support a faster CPU clock cycle and more memory (volatile and non-volatile) than their smaller siblings. High performance native controllers have a still more powerful CPU, more RAM, and maybe a removable FlashRAM card for bulk storage and backup. Like the native transmitters and actuators, these native controllers can receive power from a powered Ethernet connection or otherwise. Native controllers can be used in redundant and non-redundant configurations. Since the I/O is independent of the native controller, redundancy can be implemented in a number of manners, e.g., hardware fault-tolerance or transaction based synchronization between stations with clustering.

1.4 Native Compliant Workstations

Native workstations, e.g., such as element 40, can be constitute web browser-enabled devices that communicate with web servers, such as element 46, that actually collect the process data from other native devices. In addition, the workstations can consist of a flat panel display, OS and web browser in ROM (or on a memory card), web page (process graphic) database/cache, connections for optional annunciator keyboards, option for wireless Ethernet, and, optional, battery operation. Data supporting the operator interface comes from native devices directly.

1.5 Native Web Server

If a generic web browser-enabled device is the operator's console, a native web server, such as servers 46, 52, 56 sources the operator displays. This can be implemented with redundancy using technologies NT's clustering capabilities, or the like. Additionally, a native web server provides access to a illustrated control system 10 to users not physically attached to the illustrated control system 10, e.g., as illustrated with respect to elements 44 and 46 of Figure 1. In this mode, it centralizes data requests to maximize efficient use of communication resources.

1.6 Native Enablers

Appendix

1.6.1 The illustrated control system 10 includes native enablers 49, such as Jini and JavaSpaces devices and Solid-State DHCP servers. These enablers, which are optionally redundant, are preferably fully solid state with persistent memories. They do not use batteries for their persistent memory, though they may well plug into a wall outlet for power in non-industrial environments. The Jini and JavaSpaces serve as community "bulletin boards." These are used to notify system monitors that native devices have been added to or removed from the system. The system monitors respond to such notices by triggering appropriate actions.

The illustrated control system 10 assumes that each native device is able to acquire an IP address when it comes on-line. To minimize configuration, a DHCP server, illustrated for example by element 49, is required to furnish these addresses. To ensure maximum availability of this critical server, it is preferably a solid-state device and, optionally, includes a redundant partner DHCP Server. A particular native DHCP server provides addresses for a portion of the network 48. It obtains its configuration by notifying the system 10 that it has come on line.

Those skilled in the art will, of course, appreciate that IP addresses can be selected by mechanisms other than DHCP servers.

1.6.2 Native Internetwork Server

An internetwork server 47 is used to link separate control systems networks 48. It provides a platform for inter-network communications, controlled access to data, name conflict resolution, and other services need to efficiently, securely, and transparently accomplish the connection of one or more illustrated control systems to another.

2 SubSystems

2.1 Software Downloads

The illustrated system permits the electronic downloading of software and/or licenses for execution on the native devices. This includes native software objects, updates and/or licenses for same. To ensure proper operation of the process facility even in the event of insufficient licenses, downloaded software registers with the system monitor and declares whether or not it is

Appendix

licensed. In the case of insufficient licensing, the system monitor's notifies the customer appropriately.

In instances where appropriate native software modules and authorizations are in place, the system 10 can access a download site, e.g., an Internet e-commerce site, and determines if updates are available. If so, it notifies the user and invites him/her to request downloading of the upgrade. Such an e-commerce or other web site can also provide configuration tools that allow the user to design, implement, and test a new PCO block. This allows the user to access complex and/or rapidly improving software tools without having to maintain them locally. For this purpose, the use is charged, e.g., a modest access fee.

2.2 Security

The system 10 includes a native security system to control access to managed objects based on the type of access, e.g., (i) extra-network, i.e., users who are accessing the illustrated control system 10 from a non-native station; (ii) inter-network, i.e., users who are operating from a native workstation on a different physical native network; and (iii) intra-network, i.e., users who are operating from a native workstation within the particular native network.

Secured extra-system access is provided through a native secure web server, e.g., server 46, that permits dial-up, network, and other remote access and that supplies and defines the permitted extra-network access, including the API available to applications hosted outside the network or on the server. See, for example, elements 44 and 46 of Figure 1. Access is controlled by user name, by user location (IP address or workstation name depending on the network), and by the type of the targeted object.

Inter-system access is provided by a gateway device, such as server 47, that permits the secure transfer of data. This device negotiates secure access, deals with name conflicts between systems, and provides support for various physical media A pair of such devices are provided to account for situations where the source is local to the sink. In a preferred system, the server 47 or other gateway encrypts the data so that others cannot read it. Likewise, it authenticates message sources to verify that they are coming from a matching device. A preferred gateway

Appendix

minimizes the number of packet transfers so as to minimize delays over slow or high latency links.

Secured intra-system access is controlled based on the user and the workstation, leveraging the Java security model and other security models to the extent possible. The native security system authenticates users, e.g., regardless of whether they are attempting access from operator's console or via an application program.

The native security system manages access to objects, including PCO attributes (i.e., variables and/or parameters of PCOs), system monitors, and native messages. It allows applications to add objects to the list of managed objects. Read and write access is preferably controlled separately, i.e., as if a single PCO attribute was two separate objects.

The security model is based on a lock and key approach. Each PCO attribute, for example, is assigned to one of a large number of locks. The user is given different keys for read and write access to object attributes. This key may be modified according to the type of native workstation used for access. The key is passed to the object attribute when the object attribute is accessed. If the access is a connection, it need be passed only once. The object attribute compares the key to its lock and allow or deny access as appropriate.

A software tool is supplied with the applications development environment (ADE) or configurator that allows identification of users; access points allowed by the user; object attribute groups accessible to the user; and access type (read or write) to the object types. The object attribute groups define collections of object attributes via similar security access guidelines, e.g., alarm limits might be in one group and tuning parameters might be in another. The security model is embedded in the APIs providing access to the secured objects. This assures that that access is granted only after the native security system has performed the necessary verifications.

2.3 Maintenance and Support

The illustrated control system 10 supports the on-line upgrade of all application software from remote and local workstations. Application software, for purposes of this discussion, includes the

Appendix

system monitor, PCOs, and other Java-based code. Native diagnostic and maintenance tools work over user-supplied or other IP-based networks, providing that they allow services such as the world wide web to operate over them to external sites. For networks that do not permit this, the illustrated control system 10 provides modules that support direct connections to a native support center via dialup analog phone lines, and dialup high speed lines, e.g., ISDN and ADSL. Native maintenance, including software updates, operate over these connections; hence, it is not necessary for a person to be physically present to update system or application software.

The illustrated control system IO includes diagnostics to track problems from the hardware upwards. Maintenance staff can run in parallel diagnostic versions of the software. Such software running in open loop or switched into operation as necessary is believed particularly advantageous for support purposes.

2.4 Configuration

An application development environment (ADE) coordinates configuration in the illustrated control system 10, e.g., except for devices (such as certain transmitters and positioners) where use of an ADE is not advantageous and for which internal configuration is preferred. Characteristics of the ADE are its use in configuring/building control applications including control schemes, displays, historians, etc.; support of multiple simultaneous users; support of remote, concurrent development; support of change tracking and change management including task based activity traces, summary reports, change annotation, approval cycles, and user identification; a human interface component that runs in any web browser coupled to the system; a server component that runs on any user-supplied system; permanent storage comprising a commercial database for which a JDBC implementation is available; allowing the definition of configuration object templates; allowing the instantiation of configuration object templates into configuration objects; allowing the user to add and remove editors for configuration object components in real-time; limiting the user's access to configuration capabilities in the various pieces of equipment; and supporting application distribution along with verification of download permissions.

Appendix

With respect to the support of multiple simultaneous users, system editors provide object-based "check-out" and "check-in." No other user is allowed to edit a checked-out object or any of the objects that it contains. As an alternative, editors support the concept of task-based configuration activities and the related "check-out" and "check-in" facilities required to make it work. When an object is checked out, it is assigned to a task. Objects are not checked back in, tasks are. A build manager has the responsibility of integrating the tasks.

The native ADE delivers lower engineering and maintenance costs through the unification of application development, preservation of application expertise, reduction of application development effort, and the deployment of developed applications. Is it based on industry standards (e.g., IEC 1131 and IEC 1499) to preserve the user's investment in training and applications. It also produces appropriately formatted Java class for execution in native devices. Since the native ADE produces output that can be read and loaded into a JVM and since the native control devices include JVM, one control configurator can configure all native devices.

In a preferred embodiment, the ADE imports configurations from legacy systems; supports the use of third-party control algorithms; and supports both bulk building of control configurations and on-line changes with validation.

Configuration is not limited to implementation on a web browser, though this can be advantageous since it allows configuration from many types of device without installation of special software, i.e., it provides a thin-client configuration mechanism. For device configuration, there are two cases: a device used in illustrated control system 10 and a device used outside illustrated control system 10. For devices used outside illustrated control system 10, the configurator is placed in the device so that it can be configured even without the native ADE. This on-board configurator allows the device to be configured for use with Profibus, Foundation Fieldbus, on 4-20ma wires, and other industry-standard or proprietary networks. For devices used in the native environment, (i) a copy of the configurator is available from a native web server for off-line configuration and download, and (ii) the on-board configurator is disabled to prevent changes in the configuration except through the ADE.

Appendix

2.5 Human Interface

The illustrated system's primary human interface (HI) device is a web browser. The current range of devices executing these spans cellular phones to mainframe computers. The native HI is multilingual, i.e., it supports the presentation of information in a default language and one or more secondary languages simultaneously. All standard native applications support text substitution based on the currently selected language. Error messages are likewise in the currently selected language.

2.6 Process Control

Process Control is implemented using process control objects (PCOs) running in an execution environment consisting of a JVM and any associated applications required to load and execute the specified control strategies. Control strategies are specified in terms of PCO composites. PCOs are Java classes designed to be modular and easily upgraded even during operation. The native PCO configurator, from a high level view, creates instances of these classes, connects them as necessary, and installs them in particular devices.

Process control objects consist of two user-available types: blocks and composites. PCO blocks are, from the user's point of view, similar to conventional function blocks. Likewise, composites are similar to conventional collections of function blocks. Distinguishing external features of PCOs include the fact that changes involve the addition/deletion of PCOs and cause new versions to be loaded into the targeted native device. The new version runs in parallel with the old version until the engineer decides that it is operating properly and "swaps" the control from old to new.

Composite PCOs may span stations transparently. PCOs are bound to stations very late in the configuration process and may be migrated from one station to another at any time. Further, PCOs from different sources may operate in the same device if the configurator supports the use of multiple PCO libraries.

A native PCO configurator supports multiple libraries of PCOs from multiple vendors; permits the creation of composite PCOs from other PCOs; permits the use of composite PCOs as

Appendix

templates, with a composite PCO definition specifying which fields is altered in a template; permits the assignment of PCOs to physical devices; provides IEC 1131/1499 influenced view of the configuration process, i.e., support for Logic Diagrams, Structured Text, Sequential Function Charts, Function Blocks, and PCOs, while producing Java byte code as its output.

The creation of composites includes the raising of internal (deeply buried) names to the top level of the composite PCO. For example, a cascaded Temperature/Flow loop might have the temperature and flow measurements referenced as TC_CASCADE:PRIMARYFLOW and TC_CASCADE:SECONDARYFLOW or as the longer fully qualified name. The configurator works both off-line in a bulk creation mode and on-line in an individual correction mode.

2.7 Communications: Object Location, Message Transfer, and Data Transfer

The concepts of object location and data/message transfer are closely bound. A process control system imposes significant quality of service requirements on any services used to locate objects of interest and to acquire values from or make changes to those objects. The illustrated system includes an object location and data transfer service that provides a communication model definition, security, object location services, network types requiring support, quality of service, APIs (Java and non-Java), maintenance and upgrade strategy, and interoperability with existing control systems. In addition, the communications system addresses the particular needs of process data transfer.

2.8 Critical Applications

2.8.1 Time Synchronization

The illustrated control system 10 supports time synchronization to the millisecond in each station on the network 46. Where equipment configuration renders this impossible, time synchronization to 50 ms is provided.

2.8.2 Alarm and Message Management

The illustrated control system 10 provides a facility that centralizes viewing, recording, organizing, and categorizing the messages generated by the system monitor, the operator action journals, the PCOs, and other systems that record textual information. This application is the

Appendix

basis for alarm management strategies including inferential alarming and alarm prediction. However, the message management facility is not in and of itself an alarm historian. Rather, it relies on a native historian to provide the long-term storage of this information.

2.8.3 Historian

The native historian provides the fundamental data collection, reduction, and archival services. The data collected includes process values and messages from various applications within the illustrated control system 10. In addition, it provides historical data in support of several other common applications – typically known as plant information management system (PIMS) programs and trend window support. The historian is capable of exporting its data to user-selected databases.

2.8.4 Plant Information Management System

These applications include a simple and easy to report writing facility, a calculation facility that can use historical and real-time data to compute new values, and a desktop visualization system that uses the historians data instead of real-time data. The desktop visualization system uses the native graphics capabilities to connect the graphics to the historian and to allow the operator to "move" the entire graphic backwards and forwards through the historical data. The values calculated by the calculation facility are stored in PCOs that represent the data and generate appropriate alarms. The report writing facility supports shift reports with calendar adjustments as a simple to use feature. In addition, it facilitates the definition of ad-hoc reports using a page layout metaphor.

3 Interoperablity

Interoperability of the illustrated control system IO with legacy process control systems can be facilitated by adding a networked I/O module to the I/O chain of existing I/O cards, adding a native device integrator to talk to other networks of devices, or utilizing a native API for Microsoft- and Solaris-based applications.

4 Operating the Control System

4.1 System Startup and Management

Appendix

4.1.1 Device Identification

Each native device is assigned a name that is used to identify the device and obtain any configuration information that it needs. Embodiments of the invention use alternate approaches to identifying a device, including using a hub that is configured by the user to assign a name to each of its ports, e.g., using letterbug or software configuration; installing a letterbug, e.g., on the workshop bench, which tells the device its name; using a PC or handheld computer, e.g., on the workshop bench, to give the device its name; or using soft letterbugs.

4.1.2 Address Acquisition

After a native device has its name, it acquires its IP address from its environment. The illustrated embodiment uses DHCP for this purpose. Preferably, the DHCP server is a native device, such as enabler 49, though other DHCP servers can be used, as well.

4.1.3 Registration of the Device

Once the device knows its address, it registers its characteristics on a native bulletin board, which can be colocated with the DHCP server, e.g., in an enabler 49. Typically, many software services in the system 10 register with the bulletin board for notification of additions of native devices. Upon registration of a device, these services are notified and enter into relationships, if any, with the new device. A typical situation would involve notifying a system monitor process that a native device is active. That process then updates its web page with information about this device. In a preferred embodiment that utilizes a hierarchical network, each bulletin board is covers a given "territory" or region of the network.

4.1.4 Configuration Acquisition

If a device is unconfigured, it marks the "Configured" attribute on its bulletin board entry as "False". Any system monitor receiving device notifications to this effect generates an alert. Once a device has its name, it is able to communicate with any networked supply of non-volatile storage. This is particularly useful in the case of devices with limited in device non-volatile storage, whom a system configuration utility notifies of the location of configuration information. Devices that can retain all of their configuration information can start up using that

Appendix

configuration. Actions taken during start-up, e.g., taking PID algorithms out of hold, are configurable.

4.1.5 System Monitors

A system monitor monitors the health of system components (hardware and software), monitors the health and operation of user applications, provides configuration information to devices that request it, and monitors the licensing of software and upgrades for purposes of alerting the appropriate groups. The system monitor supports standard SNMP agents and any native specific system management protocols.

The predominant source of information used by the system monitors are the bulletin boards which, as noted above, are used by the native devices to record their current state of operation. As devices are added and removed from the bulletin boards, the system monitor displays that information. The information posted on the bulletin boards includes diagnostic information. A system monitor not only displays this information but also allows the user to access and operate any embedded diagnostic displays and operations.

4.2 Device Configuration and Management

4.2.1 Device Startup

Initial start-up of a native device involves the following steps:

1) Install the device on the network 48 and turn on the power as required by the device.
2) The device obtains an IP address from a DHCP server 49 on the network 48.
3) The device registers with the native name service.
4) The device waits for a configuration.
5) The device is now ready for configuration.

Restart of a native device is effected by the steps of

1) Install the device on the network 48 and turn on the power.
2) The device obtains an IP address from a DHCP server on the network.
3) The device registers with the bulletin board.

Appendix

4) The device begins operation according the configuration stored in its non-volatile RAM. This RAM may be a network resource that it must recover. If that resource is unavailable, the device is essentially unconfigured and the above procedure is followed.
5) The device is now ready for configuration and operation.

4.2.2 On-Line Configuration

The steps involved in on-line native device configuration are:

1) The engineer starts a web browser on workstation 40, handheld computer 44 (wireless or otherwise), a PC, a native workstation, etc.
2) The default home page on the browser is cached and contains an applet that locates the native bulletin board and raises the initial web page that lists the services available from that workstation.
3) The engineer selects the ADE and a native device of interest.

If the device is on a native network 48, the native configurator (ADE) is used to make the changes. In this case, the web browser is pointed at the web server that hosts the ADE. This approach eliminates conflicts between field change and database changes. The configuration of non-native devices is provided by a service that encapsulates the native commands and passes them through a native device to the actual foreign device.

The steps in PCO configuration are

1) As in the off-line mode, the engineer accesses the ADE and makes necessary changes.
2) Once the changes are complete, the engineer tells the new objects to begin execution in "open loop" mode, i.e., their outputs are not allowed to go to the field or to any other display station other than the one used to configure them. (For examples, their names are not registered with the native name service.)
3) In the open loop mode, the engineer can "Verify and Validate" his new control scheme by viewing the detail displays for the new and the old objects "in parallel". The set of available detail displays may include special displays for evaluation of parallel composites.

Appendix

4) Once the configuration is complete, the device records its new configuration in its local non-volatile RAM. Optionally, it updates a remote configuration database or just set a "changed configuration" status indicator much like conventional intelligent transmitters.

5 Native Software Subsystems

5.1 Security

5.1.1 Object Access

5.1.1.1 Locks and Keys

As noted above, the proposed security mechanism is a lock and key. In this model, the attributes of an object that are available to be manipulated are assigned a lock number. An application registers with the security system at start-up to obtain a key. The value of the key depends on the effective user id (name) and the station on which the application is running. The key is never visible to the user, so he/she cannot alter it. This key is available to any library that uses the security system to ensure proper access.

5.1.1.2 Security Database

A security database contains a list of users. For each user it allows the specification of a master key, i.e., a bit array with a bit set for each lock the user is allowed to unlock. There is one master key for each object type supported by the security system. By default, the security system supports the creation of master keys for the system monitor, process data (PCO attributes), and native messages (events). The database also allows modifications to the master key for a user based on the station used to perform an operation. These exceptions can be expressed as IP addresses, station names, or by the type of access being used (e.g., extra-system, inter-system, or intra-system).

5.1.1.3 Operation

The security system supports caching the security system database on specific servers to improve redundant operation and to speed key queries. This is generally not a significant issue since the query is done once per application. Each time an operation is attempted or, in some circumstances, once per connection, the key is passed to the target object along with any other information required by the object to achieve the user's request. It is the responsibility of the object to match a bit in the key with the lock on the attribute or method being altered. If there is a

Appendix

match, the operation proceeds. If the match fails, the operation is aborted and the calling application is notified of the failure. If the lock is zero, the operation is allowed to continue.

5.1.1.4 Use

According to a preferred practice, the illustrated control system 10 supports at least 256 individual locks. This allows the application engineer to define 255 groups of objects for each type of object being accessed.

5.1.1.5 Impact on PCOs and Other System Components

The value record of a PCO attribute includes two keys: one for read access and one for write access. A default value for the key can be inherited according to rules concerning the use of the attribute, e.g., all alarm attributes have an assigned group by default. A system monitor interface implements the same mechanism.

5.2 Maintenance and Support

5.3 Configuration

A configuration object (CO) is a collection of objects plus the editor used to manipulate them. A Configuration Object Template (COT) is a CO that has not been instantiated. The application development environment (ADE) is the environment for configuration object editors. It shows the user the existing COs in the CO being edited and a list of available templates. It provides the mechanism for adding objects to a CO by instantiating a COT and for removing COs. A Configuration Object Component (COC) is any of the objects found in a Configuration Object. Typical components are: PCOs, Process Graphics, Reports, Historian Configuration Objects, etc. The ADE has two components: the human interface and the server. Several human interfaces can access the same server at any time.

When a Native device is placed on-line, its internal configuration services are disabled. All configuration instructions come from the application development environment. Where on-line configuration is permitted, the system monitor or other system component flags any configuration mismatches.

Appendix

Invoking the ADE is tantamount to starting the editor of a particular top-level CO. The ADE presents all of the available COs within the selected CO and all of the available COTs. The ADE provides a mechanism for manipulating those objects and for adding new objects by instantiating COTs. Since the objects within the ADE are also Configuration Objects, i.e., they are themselves collections of objects with associated editors, configuration involves selecting an object, invoking its editor, and making changes. Each editor knows how to store its object's data in the persistent storage.

The ADE preferably includes a system editor, a control editor, a graphic editor, a message manager editor, a historian editor. All editors support the use and creation of templates by the user. The also support the assignment of their objects to particular stations in the native network. Not all objects require assignment, but the editors are responsible for it.

5.3.1 System Editor

The system editor allows the user to configure a logical arrangement of native device objects.. The only devices it configures are those that are native-enabled. If a device, such as Networked I/O, does not support Native, this editor is not interested in it. The following table lists those devices, what they do, and what needs to be configured.

Device Type	Description	System Editor Configured Attributes
DHCP/Bullet in Board Server	These devices allocate IP addresses to stations and act as the bulletin board for the stations that ask for addresses. They are not configured to know what devices might be attached.	Name and association with a Web Server
Native Controllers	These devices act as PCO platforms.	Name and association with a DHCP/bulletin board Server (which may be a Web Server)
Native Workstations	The devices that supply the human interface to the process.	Name and association with a DHCP/bulletin board Server (which

Appendix

		may be a Web Server)
Native Web Servers	This devices host Native Applications and graphics. Native Workstations may also be Web Servers.	Name

As with the other editors, the system editor is object oriented and template driven. The user is allowed to select an object type from a palette and attach it to the appropriate stations. Individual Device Editors are provided for each of the device types.

5.3.1.1 DHCP/Bulletin Board Editor

If the device is assigned to another DHCP/bulletin board device or a web server, the editor allows the user to specify the number of IP addresses that this device needs to be able to supply below it. In this case, when the device starts up, it gets its own IP address and IP address range from the Web Server or DHCP/bulletin board device to which it is assigned. If it is not assigned to such a device, the editor allows a range of IP addresses to be assigned to it. It is assumed that the device gets its own IP address from another, non-Native DHCP Server.

5.3.1.2 Native PCO platforms

Each device type on the network is provided a specific editor to control its specific configuration. Typical devices are: transmitters, actuators, controllers, etc. These editors are available for off-line configuration. In this case, the system monitor downloads any approved configurations at the next start-up. The editors may also be available on-board the device (e.g., for fixed function devices such as single station controllers), which allows direct configuration through a web browser. The on-board facility is automatically disabled once the device is placed on a native network. This ensures database consistency while allowing the use of the device on non-native networks. In the illustrated embodiment, device specific configurators are provide in ADE add-in and on-board, preferably, with the same code would be available and used for both.

5.3.1.3 Native Workstations

Appendix

The configuration of this device includes user name(s) allowed; read only operation; full fledged operator; and so forth.

5.3.1.4 Native Server

The configuration of this station is a range of addresses if it is acting as a DHCP server.

5.3.2 The Control Editor

The Control Editor implements PCOs plus four of the IEC 1131-3 languages (LD, ST, SFC, and FB). Any control schemes built in those languages are translated into a Java file, compiled, and downloaded. The IEC 1131-3 concept of configurations, resources, programs, and tasks are translated into the native environment that consists of set of native platforms. In addition, the Control Editor supports defining of control schemes using PCOs in the same manner that it supports the use of IEC Function Blocks; supports the assignment of control entities to specific PCO platforms; the importation of instruments lists to generate, and PCOs.

5.3.3 The Graphic Editor

The Graphic Editor support all of the human interface (HI) functionality and graphics typical of this type of editor.

5.3.4 Application Configuration

Applications can only be assigned to native web servers.

5.3.4.1 Historian Object Editor

A historian object when found in a CO is a representation of a portion of the full historian configuration, e.g., a slice of the historian. The historian object consists of the name of the historian to receive the data and a list of PCO attributes. Each attribute is associated with certain critical information related to the historian, e.g., change deltas, sample rates, reduction operations, archive characteristics, etc. Starting the historian editor resulting in showing the full configuration of the historian(s) to which this object is assigned.

5.3.4.1.1 Historian Data Definition Object Editor

Appendix

One historian object exists for each historian in the system. These objects are created automatically if a historian data definition object is created and assigned to a historian. The data in this object controls the general function of the historian, e.g., where it runs in the system, what its archive policy is, what its data reduction policy is, etc.

5.4 Process Control

5.4.1 Definition of the Process Control Domain

The process control domain is usually divided into three primary control styles: (i) analog control (continuous), (ii) logic control (ladder logic), and (iii) sequential control (supervisory operation of logic and analog control structures). In addition, every process control system must address batch (or recipe based) manufacturing with the related topics of lot tracking and validation.

The illustrated control system 10 views these styles as particular visualizations of the process control functionality. This means that these three types of control do not exist as objects. Rather, the control system receives a class definition of a process control object (PCO) which it instantiates and executes. It is the responsibility of the control editor to create the correct representation of the control structure while also representing that control structure in one of the industry standard formats, i.e., ladder-logic, function blocks, or structured text, or PCOs directly

Sequential Function Charts are a meta-language used to control the execution of the other three languages. In the illustrated embodiment, it serves as a wrapper class for the other classes. The source code for a control scheme and the critical values for its PCO attributes, i.e., its checkpoint file, are held in the non-volatile memory of the Native device.

5.4.2 Process Control Objects

5.4.2.1 PCO Attributes

The fundamental control object is a PCO attribute. As noted above, these are provided in two types: variables and parameters. An example of a variable attribute is a measurement in a PID block. In the illustrated system, a PCO attribute is an object in its own right and it contains certain vital information when it is retrieved. This information is "atomic", i.e., all information is obtained with an attribute, not just its "value". The minimal set of information from a PCO

Appendix

variable attribute is value, data type, quality information time tag in milliseconds, and alarm status. It is this object that is transferred from one PCO to another to implement the transfer of information needed in an actual control scheme.

Parameter attributes bundle less information than a variable attribute. The typical example of a parameter attribute is the proportional band of a PID block. In the illustrated control system 10, this attribute has less supporting information than a variable attribute, e.g., it would not have an engineering unit range.

5.4.2.2 PCO Blocks

PCO attributes and methods to operate on those attributes are combined into PCO Blocks. These blocks are analogous to the control blocks in other systems, e.g., I/A Series FoxBlocks, IEC 1131-3 Function Blocks, and Foxboro Microspec Blocks. However, because they are implemented in Java, they can be executed on any microprocessor running any OS that supports a JVM. In addition, they are designed to occupy only the amount of RAM required for the selected options and to be on-line replaceable.

PCO blocks capture and encapsulate knowledge of regulatory control algorithms needed to by continuous processes. The PCO user block can be used to integrate particular user-defined control algorithms into a PCO control scheme. It is the PCO user block that is used to implement control strategies that are defined using one of the IEC 1131-3 control languages.

5.4.2.3 PCO Composites

PCO Composites are implementations of a control strategy. They consist of PCO blocks and other PCO composites linked in such a manner as to provide the control mechanisms needed for a particular process. It is the PCO composite that captures application level control expertise, e.g., complex control structures for major pieces of equipment or even the simpler idioms of cascade control.

5.4.3 PCO Execution Environment

Appendix

The PCO execution environment is the software in a native control device that supports the execution of PCOs. This software includes the JVM and any other required code, such as a class loader that loads stand-alone applications for each composite, a class loader that loads an applet for each composite, a block processor that loads Java byte codes for a composite and executes each composite in turn, and a block processor that loads Java byte codes for PCO Blocks and executes each blocks in turn.

5.5 Communications

5.5.1 Requirements

5.5.1.1 Communication Model Definition

Several types of communication are provided in the illustrated control system 10. These types include data transfer, message transfer between application programs, object movement (downloads), class file (code) transfers, and events (unsolicited messages).

The communications mechanism is redundant, i.e., the failure of one station in the system does not prevent communication (new or established) between any two other stations. It also minimizes the impact on uninvolved stations when the service is used by its clients, e.g., a message is not sent to a network segment if it is not going to be used there. The communications mechanism, moreover, provides an encapsulated implementation that can be replaced "under the hood" as new technologies become available. Also, it allows the illustrated control system 10 to be configured for the appropriate communication resources, e.g., the number of outbound connections. Moreover, it provides performance statistics, e.g., high, low, average transfer times between two stations, bytes per seconds between two stations, etc., for viewing in the system monitor. Still further, it automatically reconnects to an object if it is relocated on or removed from/restored to the network.

Data transfer in the illustrated control system 10 is provided by a mechanism that permits clients to subscribe to updates in a object. It also provides a mechanism for one-shot get/set access to an object. A client of the data transfer facility supplies the control system with a (i) number of objects that it needs, (ii) conditions under which it wants updates: change delta, maximum time between updates, or both, (iii) be capable of retrieving the values all at once (a snapshot), and

Appendix

(iv) be capable of retrieving the values as series of changes with the option to block, for a user specified amount of time, while waiting for a change. Client publication (write lists) are not supported in the illustrated embodiment. Target devices subscribe to the data in the source device.

With respect to client one-shots (get), the client of the data transfer facility supplies the system with a number of objects that it needs and retrieve the values all at once (a snapshot). Conversely, with respect to client one-shots (set), the client supplies the system with a number of objects that it wishes to change, and set any part of the object if it has access. The parts referred to are the supporting information in a PCO attribute as well as the value part.

5.5.1.1.1 Message Transfer

5.5.1.1.1.1 General

While the message transfer mechanism is a general case of the data transfer mechanism, its requirements are given separately. The client of the message transfer facility must support private conversations between applications using a reliable communications mechanism and support by subscription publication of information from one server to many clients.

5.5.1.1.1.2 Private Conversations

The private conversation mode allows applications to find each other by their application name: the client application is not required to know the station name or IP address of the server application. It also allow applications to send records (messages) of arbitrary size: the private conversation mode guarantees that the message boundary is recognized and that order is preserved. Moreover, it allows both the sending and receiving process to determine if the connection is lost. Still further, it supports transaction based message transfer, i.e., the sender does not get an acknowledgement until the receiver says that the message is safely stored.

5.5.1.1.1.3 Subscriptions

The subscription facility supports senders and listeners. A sender is a process that has data that may be of interest to one or more listeners. The subscription mode supports this type of operation by allowing a "sending" process to register that it will produce messages of a particular type,

Appendix

allowing a "listening" process to register to receive all messages of a particular type, supporting transaction based subscription if the client so requests (this request would have no impact on the sender), and supporting transaction based publishing by a server if the server requests it (this request would have no impact on the receiver). If no listeners are registered, the subscription facility defines how the message is handled.

5.5.1.2 Security

The native communications system is linked to the native security system to ensure that unauthorized users are unable to access objects. The illustrated control system 10 does not arbitrate between users of a resource. The prevention of multiple access to an object is the responsibility of the object. The breaking of an application secured link by an operator is not a function of the illustrated control system 10.

5.5.1.3 Object Location Services

Object are located by name. To this end, the object location service supports the location of named objects; supports hierarchical object names; allows rule based specification of the name delimiting character; locates an object based on a "longest fit" because (a) not all parts of an object name are globally known and (b) not all parts of an object are in the same physical location; supports the implementation of naming scopes, i.e., limiting the visibility of names; supports the use of a name search path so that relative names can be located; is redundant; supports locating many types of objects, e.g., server processes, PCO attributes, and application programs; and supports the addition of related information about the object in the name directory, e.g., object type, short-cut information to be used by the process that owns the object, object type/implemented interface, and information. With respect to the last item, the object location services permit a client to request the name of a service that supports a specific interface. This allows new interfaces to be added to a service without breaking an old one.

5.5.1.4 Networks

The illustrated control system operates over any IP based network 48. By way of non-limiting example, the illustrated control system 10 operates over wide area networks supplied by the customer, installed plant LANs supplied by the customer, dedicated LANs supplied either by the

Appendix

customer or the vendor, redundant LANs supplied by the vendor. This operation is transparent to all applications using the system 10. While it is recognized that performance are degraded over low bandwidth (phone lines) and high latency (geosynchronous satellites) networks, the system 10 optimizes as best it can to minimize the impact of different network types. In short, the illustrated control system 10 does not assume that it "owns" the network.

5.5.1.5 Quality of Service

5.5.1.5.1 Performance

Assume that the illustrated control system 10 is operating in support of a PCO in a dedicated control station (not a transmitter) or in a PC, it supports at least 10,000 object value gets/sets per second if the object is in the local station; supports at least 100 get/set per second requests from a client if the object is in a remote station; supports at least 50 get/set requests per second made to a server from a remote client; is able to locate 3000 objects per second; is able to detect a communication failure in no more than 1000 milliseconds; ensures that the time to move data from a server station to a client station (API call to API call) is less than 100 milliseconds in a normally configured network; allows a server station to move 6000 values out of its box every second; allows a client station to accept at least 2000 values every second; allows an unlimited (but, configured) number of names to be defined in the system; and supports the registration of 50,000 globally known names without requiring a reconfiguration. All server stations are able to support at least 30 client stations for continuous data feed without requiring a reconfiguration. All client stations support at least 30 server stations for continuous data feeds without requiring a reconfiguration.

5.5.1.6 APIs

Within the illustrated control system 10 environment an API are provided for all of the above services. This API are delivered as a Java Bean so that the underlying communications mechanism may be replaced without breaking all of the applications that use the service.

5.5.1.7 Maintenance and Upgrades of Messaging Services

The illustrated control system 10 defines and enforces a mechanism for supporting inter-operability of messages from stations at different versions. A mechanism like interface

Appendix

inheritance is provided, i.e., a message carries in it enough "type" information that the receiving application knows how to "decode" the message even if the receiving application is at a lower level. This adds fields to messages if necessary, but not remove/replace old fields.

5.5.1.8 Integration of non-Native Based Systems

The illustrated control system 10 supports integration of non-Java applications using appropriate technologies, e.g., CORBA, COM/DCOM/ActiveX, and emulation of APIs. Physical gateways are permitted, but these gateways provide only a minimal amount of configuration.

5.5.1.9 Object Location Service Implementation

5.5.1.9.1 The All Java Approach

5.5.1.9.1.1 Description

JINI and JavaSpaces technologies permit object location within the illustrated system. JINI technology allows native devices to register names in a globally available database. JINI clients can find this database and perform searches against it. JavaSpaces allow a device to register important information in a globally available bulletin board. JavaSpaces provide four simple services: posting, removal, query by example, and notification. The notification feature informs prospective users of a class of objects when such objects are added and removed from the JavaSpace. A JavaSpace handles services like system monitoring.

5.5.1.9.2 The Enhanced Object Manager Approach

5.5.1.9.2.1 Description

As an alternative, an enhanced object manager (EOM) approach to address location and connection of data clients and servers can be utilized as follows:

1) A LDAP compliant service are created and used to store pathnames and related information, e.g., IP addresses, Port Numbers, internal indexes, etc.
2) The EOM API provides get/set and read/write support similar to that provided by the OM today.
3) When provided a name, the EOM looks in the local copy of the LDAP database.

Appendix

4) If the name is found and the data is local, the request is passed to the correct data provider (the EOM, the CIO database, or the AOS database) who returns the value and other data. The EOM handles sending updates on change driven data.
5) If the name is found and not local, the EOM sends the request to the remote EOM which then makes a local query. The EOM handles sending updates on change driven data.
6) If the name is not found locally, the location request is passed to the master LDAP database. If it is found there, the address information is passed back to the local LDAP database (which is updated) and steps 4) and 5) are repeated.
7) If the name is not found remotely, the request is posted for a period of time. If it is resolved in that period, the requesting station is notified. If it is not resolved, the master LDAP database drops the name and notifies the requesting station.

5.6 Critical Applications

5.6.1 Time Synchronization

SNTP (Simple Network Time Protocol) is used to keep operator stations and similar non-process data producers in sync. This protocol is as accurate as 1 ms in a carefully controlled environment, but 50 ms is more common on shared networks. Controllers requiring high accuracy are placed on tightly controlled networks or equipped with an interrupt used to coordinate time updates. In either case, a master timekeeper tied to a well known reliable data source - GPS clock or broadcasted time from the national atomic clocks. This master time keeper generates the interrupt and SNTP. It is built with a highly reliable internal clock so that loss of time feed does not result in significant drift within the Mean Time to Repair (MTTR).

5.6.2 Alarm and Message Management

Alarm and message manage requires that all process alarming to be based on the top-level composite's attributes; that acknowledgments be handled at the composite level; that PCOs, system monitors, and the native human interface a reliable message protocol to send the message to a message management system (MMS); that the MMS is used by end-users to view the current alarm state and the alarm history; and that the MMS use the plant historian to permanently store and archive messages.

Appendix

To meet the control market's requirement for a message management system, the illustrated control system 10 provides a message management support service (MMSS) in the native devices that are used to deliver messages to a Message Manager. The service operates as follows.

When a client of the MMSS connects to the MMSS server, the MMSS server send any messages in its internal alarm queue that are older than the time of the connection to the client; transmits the alarm state at the time of the connection to the client, and transmit any new messages subsequent to the connection. If the connection to the MMSS client fails, there is no impact on the operation of the control station or the control network; minor delays associated with detecting the loss of connection are generally viewed as acceptable. The MMSS support at least two (2) and, preferably, four simultaneous clients.

The alarm state of a PCO attribute is defined to include block attribute identification (full PATH attribute name, at least); the bad I/O status of the block for I/O attributes; an indication of the alarm status for each alarm type defined for the attribute; the priority, criticality, and acknowledgment status of the attribute; the on/off status of the attribute, i.e., is it updating; the status of the alarm inhibit and alarm disable or alarm option parameters, and the time tag of each alarm event stored within the block, e.g., the into alarm time, the acknowledged time, and the return to normal time. This information is recovered from all currently active alarms.

5.6.2.1 Configuration

5.6.2.1.1 Process Alarms

When a composite is configured, the application engineer specifies which attributes of the PCOs within the composite are to be annunciated. When a composite is placed on line, it notifies the object location service that it exists. A message management service are notified of the new composite and arrange for notification from the device.

5.6.2.1.2 System Alarms and Messages

No user configuration is required to get these alarms into the MMS. The sources of such messages arranges for the appropriate notification.

Appendix

5.6.2.1.3 Native Operator Actions

No user configuration is required to get these alarms into the MMS. The sources of such messages arranges for the appropriate notification.

5.6.2.2 Native Message Management Service

The MMS are centralized and redundant. It provides a complete alarm state for the plant at any time including startup. It retains a message history for as long as configured through the use of the plant historian. Users view the alarms through to the Alarm Manager.

6 Native Hardware SubSystems

6.1 Field Devices

6.1.1 Overview

The native field devices are coupled to the network 48 via an IP network, preferably, Ethernet and, still more preferably, powered Ethernet. Powered Ethernet delivers the power, along with conventional Ethernet data. To this end, the native field devices use one of two wiring schemes, with each scheme supplying power to the field device: (a) standard four wire 10Base-T wiring, or (b) industry standard two-wires used for 4-20ma based transmitters. The devices also provide a single Ethernet interface (or redundancy, where desired in the cabling or the field device); and connect to a non-redundant powered hub with the following features: (a) non-redundant connections to the IFDs; and (b) support for redundant connection to a plant-wide network. Where used, the powered Ethernet implementation leaves the physical and logical Ethernet interfaces in place. This means that except for adding a new wiring type, the physical layer, the data link layer, the use of CSMA/CD, and 10Base-T connectors are preserved.

The illustrated embodiment utilizes powered Ethernet hubs to provide communications and power to its intelligent field devices. These IFDs may be transmitters, actuators, Networked I/O, or PCO platforms. The powered hubs are preferably, stackable, DIN rail-mountable, support connection to a redundant network, and support both 10 Mbps and 100 Mbps Ethernet.

Appendix

A further understanding of the operated of the powered Ethernet network and of the circuitry used within the native field and control devices to draw power therefrom may be attained by reference to United States Patent Application Serial No. 09/444,827, filed November 22, 1999, entitled POWERED ETHERNET FOR INSTRUMENTATION AND CONTROL, and by reference to the corresponding PCT Patent Application US00/10013, filed April 14, 2000, the teachings of which are incorporated herein by reference, and a copy of which is attached as an appendix hereto.

On the software side, specific implementations of the native field devices support web based configuration, as described above. The support is supplied by including an embedded web server with a selection of pages used for configuration and maintenance.

6.1.2 Transmitters

Intelligent transmitters use the IEEE 1451 standard to communicate to their sensor(s). This facilitates use of the same module in many types of transmitters from potentially many vendors and allows the transmitter to support up to 255, perhaps externally mounted, sensing devices.

6.1.3 Positioners

The native intelligent positioners also support a powered Ethernet interface like the one used by the native intelligent transmitters. Figure 6 shows a preferred native intelligent positioner implementation. The approach standardize interfaces and use a PCO to control the positioner as evident in the drawing.

6.2 Summary of Hardware/Software Features by Station Type

A summary of the features of native control devices follows. Blank cells indicate that the feature is not present in the indicated device.

Features	Xmitter	Positioner	Controller	Integrator	PC based Operator Consoles	Native Workstation	PC Based Web Server	Solid-state Web Server	SS DHCP Server	SS bulletin board
Low power, high performance CPU	Yes	Yes				Yes				
Top performance CPU			Yes	Yes	Yes	Yes	Yes	Yes		
RAM	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ROM	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Flash	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes
Sensor I/F (IEEE 1451)	Yes	Yes								
Powered Ethernet (2 wire)	Yes	Yes								
Serial I/F (RS-232/485/422/423)	Yes	Yes		Yes						
Ethernet (10Base-T)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

- 43 -

Appendix

Appendix

Features	Xmitter	Positioner	Controller	Integrator	PC based Operator Consoles	Native Workstation	PC Based Web Server	Solid-state Web Server	SS DHCP Server	SS bulletin board
Wireless Ethernet						Yes		Yes		
the TCP/IP suite	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DHCP Server							Yes	Yes	Yes	Yes
Full OS	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Java Virtual Machine	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Web Server	Yes	Yes	Yes	Yes		Yes	Yes	Yes	Yes	Yes
Device Configurator	Yes	Yes	Yes	Yes					Yes	Yes
Annunciator					Yes	Yes				
Keyboard										
Touchscreen					Yes	Yes				
Mouse/Trackball					Yes	Yes				
Display					Yes	Yes	Yes	Yes		
Hard drive					Yes		Yes			
Solid State Bulk Storage						Yes		Yes		
System Monitor							Yes	Yes		
Process Control	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		

- 44 -

Appendix

Features	Xmitter	Positioner	Controller	Integrator	PC based Operator Consoles	Native Workstation	PC Based Web Server	Solid-state Web Server	SS DHCP Server	SS bulletin board
Objects (PCOs)										
System Management Objects	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Omnibus Messaging Services	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes		
Profibus PA Support	Opt.	Opt.		Yes						
Profibus DP Support	Opt.	Opt.		Yes						
HART Support	Opt.	Opt.		Yes						
FoxComm Support	Opt.	Opt.		Yes						
Fieldbus HSE Support	Opt.	Opt.		Yes						
Fieldbus H1 Support	Opt.	Opt.		Yes						
DeviceNet	Opt.	Opt.		Yes						

Appendix

Features	Xmitter	Positioner	Controller	Integrator	PC based Operator Consoles	Native Workstation	PC Based Web Server	Solid-state Web Server	SS DHCP Server	SS bulletin board
LONworks	Opt.	Opt.		Yes						
Modbus				Yes						
Modbus+				Yes						
ABDH				Yes						
I/A Series Nodebus				Yes						
I/A Series Fieldbus (FoxComm)	Opt.	Opt.		Yes						
Application Development Environment							Yes			
Historian							Yes			
Time Synchronization							Yes	Yes		
PIMS							Yes			
Message Management							Yes			

Appendix

Described above are apparatus and methods that achieve the goals of the invention. It will be appreciated that the embodiments described herein are examples and that other embodiment incorporating changes thereto also fall within the scope of invention. Thus, by way of example, it will be appreciated that the invention can be implemented using a virtual machine environment other than JVM and using bytecode other than Java bytecode. By way of further example, it will be appreciated that the apparatus and methods taught herein can be applied to a range of control application, in addition to process control.

In view of the foregoing, what I claim is

Appendix

1. A field device for a control system, the improvement wherein

the field device provides a virtual machine environment and executes byte code therein,

the byte code configuring the field device to execute a control algorithm.

2. The field device of claim 1, the further improvement wherein the byte code configures the field device to execute a control function block.

3. The field device of claim 1, the further improvement wherein the byte code configures the field device as a controller.

4. The field device of claim 3, the further improvement wherein the byte code configures the field device to perform proportional integral derivative control.

5. A field device for a control system, the improvement comprising

a virtual machine environment within the field device,

byte code executing within the field device that configures it to perform signal conditioning.

6. The field device of any of claims 1 - 5, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

7. The field device of any of claims 1 - 5, the further improvement wherein the byte code comprises JAVA byte code.

8. A field device for process control, the improvement wherein the field device comprises a web server.

Appendix

9. The field device of claim 8, the further improvement wherein the web server is embedded.

10. The field device of claim 8, the further improvement wherein the web server serves web pages that facilitate any of configuration, monitoring and maintenance of at least the field device.

11. The field device of claim 10, the further improvement comprising a configuration editor in communication with the web server, the configuration editor controlling at least the field devices' configuration.

12. The field device of claim 11, the further improvement wherein the configuration editor is selectively disabled or enabled, depending upon the type of network to which the field device is coupled.

13. A field device for a control system, the improvement wherein the field device provides a web server and a virtual machine environment.

14. The field device of claim 13, the further improvement wherein the virtual machine environment executes byte code that configures the field device to any of (i) execute a control algorithm and (ii) perform signal conditioning.

15. The field device of claim 14, the further improvement wherein the byte code configures the field device as a controller.

16. The field device of claim 15, the further improvement wherein the byte code configures the field device to perform proportional integral derivative control.

17. The field device of claim 14, the further improvement wherein the web server is embedded.

Appendix

18. The field device of claim 14, the further improvement wherein the web server facilitates any of configuration, monitoring and maintenance of at least the field device.

19. The field device of claim 18, the further improvement comprising a configuration editor that is in communication with the web server, the configuration editor controlling at least the field devices' configuration.

20. The field device of claim 19, the further improvement wherein the configuration editor is selectively disabled or enabled, depending upon the type of network to which the field device is coupled.

21. The field device of any of claims 13 - 20, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

22. The field device of any of claims 13 - 20, the further improvement wherein the byte code comprises JAVA byte code.

23. A field device for a control system, the improvement wherein the field device comprises an interface to an IP network.

24. The field device of claim 23, the further improvement wherein the interface receives operational power for the field device from the IP network.

25. The field device of claim 23, the further improvement wherein the interface comprises an interface to an Ethernet network.

26. The field device of claim 25, the further improvement wherein the interface receives operational power for the field device from the Ethernet network.

Appendix

27. The field device of claim 23, the further improvement comprising a processor that is in communication with the interface.

28. The field device of claim 27, the further improvement wherein the processor is a low-power processor.

29. The field device of claim 27, the further improvement wherein the processor provides a virtual machine environment and executes a web server.

30. The field device of claim 29, the further improvement wherein the processor executes an operating system.

31. The field device of claim 30, the further improvement wherein the operating system is a real-time operating system.

32. The field device of claim 30, the further improvement wherein the field device comprises at least one of a random access memory, a read-only memory, FlashRAM, a FlashRAM, and a sensor interface.

33. The field device of any of claims 23 - 32, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

34. A field device for process control, the improvement comprising

a processor, and

an interface to an IP network, the interface being in communication with the processor.

35. The field device of claim 34, the further improvement wherein interface receives operational power for the field device from the IP network.

Appendix

36. The field device of claim 34, the further improvement wherein the interface comprises an interface to an Ethernet network.

37. The field device of claim 36, the further improvement wherein the interface receives operational power from the Ethernet network.

38. The field device of claim 34, the further improvement wherein the processor executes a web server.

39. The field device of claim 34, the further improvement wherein the processor executes any of (i) a control algorithm and (ii) a signal conditioning algorithm.

40. The field device of claim 39, the further improvement wherein the processor performs proportional integral derivative control.

41. The field device of claim 34, the further improvement wherein the processor executes a web server that facilitates any of configuration, monitoring and maintenance of at least the field device.

42. The field device of claim 41, the further improvement comprising a configuration editor in communication with the web server.

43. The field device of claim 42, the further improvement wherein the configuration editor is selectively disabled or enabled, depending upon the type of network to which the field device is coupled.

44. The field device of any of claims 34 - 43, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

Appendix

45. The field device of claim 44, the further improvement wherein the field device is adapted for use in a process control system.

46. A field device for a control system, the improvement wherein the field device includes an interface to an IP network and wherein the field device issues to the IP network, via the interface, a request for assignment of an IP address.

47. The field device of claim 46, the further improvement wherein the field device receives a device identification name from any of (i) a user-configured hub or other device to which the field device is coupled, (ii) a letterbug installed in the field device, (iii) a digital data processing apparatus, (iv) a software generated letterbug.

48. The field device of claim 46, the further improvement wherein the field device registers a characteristic via the IP network.

49. The field device of claim 48, the further improvement wherein the field device registers the characteristic with a bulletin board on the IP network.

50. The field device of claim 49, the further improvement wherein the field device registers the characteristic with in a Javaspace on the IP network.

51. The field device of claim 46, the further improvement wherein the field device communicates with another element of the system over the IP network in order to obtain configuration information.

52. The field device of claim 46, the further improvement wherein the field device retains configuration for use at startup.

53. The field device of any of claims 46 - 52, the further improvement wherein the field device has a processor that performs any of (i) a control algorithm and (ii) signal conditioning.

Appendix

54. The field device of claim 53, the further improvement wherein the processor configures the field device as a controller.

55. The field device of claim 54, the further improvement wherein the processor configures the field device to perform proportional integral derivative control.

56. The field device of any of claims 46 - 52, the further improvement wherein the field device includes a web server that facilitates any of configuration, monitoring and maintenance of at least the field device.

57. The field device of claim 56, the further improvement wherein the field device comprises a configuration editor.

58. The field device of claim 57, the further improvement wherein the configuration editor is selectively disabled or enabled, depending upon the type of network to which the field device is coupled.

59. The field device of any of claims 46 - 52, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

60. The field device of claim 59, the further improvement wherein the field device is adapted for use in a process control system.

61. A control device for a control system, the improvement wherein

the control device provides a virtual machine environment and executes byte code therein,

the byte code configuring the control device to execute a control algorithm.

Appendix

62. The control device of claim 61, the further improvement wherein the byte code configures the control device to execute a control function block.

63. The control device of claim 61, the further improvement wherein the byte code configures the control device as a controller.

64. The control device of claim 63, the further improvement wherein the byte code configures the control device to perform proportional integral derivative control.

65. The control device of any of claims 61 - 64, the further improvement wherein the control device comprises any of web server, control station, operator console, personal computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

66. The control device of any of claims 61 - 64, the further improvement wherein the byte code comprises JAVA byte code.

67. A control device for a control system, the improvement wherein the control device provides a web server and a virtual machine environment.

68. The control device of claim 67, the further improvement wherein the virtual machine environment executes byte code that configures the control device to execute a control algorithm.

69. The control device of claim 68, the further improvement wherein the byte code configures the control device as a controller.

70. The control device of claim 69, the further improvement wherein the byte code configures the control device to perform proportional integral derivative control.

Appendix

71. The control device of claim 67, the further improvement wherein any of the web server and the virtual machine environment is embedded.

72. The control device of claim 67, the further improvement wherein the web server any of (i) facilitates any of configuration, monitoring and maintenance of the control system or one or more control devices, (ii) collects process data from one or more control devices, (iii) generates source for operator displays, (iv) provides access to the control system, and (v) hosts an applications development environment.

73. The control device of any of claims 67 - 72, the further improvement wherein the control device comprises any of a web server, control station, operator console, personal computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

74. The control device of any of claims 67 - 72, the further improvement wherein the byte code comprises JAVA byte code.

75. A control device for a control system, the improvement wherein the control device comprises

a low-power processor,

an interface to an IP network, the interface being in communication with the processor, and

the interface receives operational power for the control device from the IP network.

76. The control device of claim 75, the further improvement wherein the interface comprises an interface to an Ethernet network.

Appendix

77. The control device of claim 75, the further improvement wherein the processor provides a virtual machine environment and executes a web server.

78. The control device of claim 77, the further improvement wherein the control device comprises at least one of a random access memory, a read-only memory, FlashRAM, and a sensor interface, access to permanent storage, a configurator, system management software, messaging services, alarm/event notification, byte code implementing process control functions, byte code implementing status reporting.

79. The control device of any of claims 75 - 78, the further improvement wherein the control device comprises any of a web server, control station, operator console, personal computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

80. A control device for process control, the improvement wherein

the device comprises an interface to an IP network, and

on startup, the device registers a characteristic thereof with at least one other device on the IP network.

81. The control device of claim 80, the further improvement wherein the control device issues to the IP network, via the interface, a request for assignment of an IP address.

82. The control device of claim 80, the further improvement wherein the control device receives a device identification name from any of (i) a user-configured hub or other device to which the control device is coupled, (ii) a letterbug installed in the control device, (iii) a digital data processing apparatus, (iv) a software generated letterbug.

83. The control device of claim 80, the further improvement wherein the control device registers the characteristic with a bulletin board on the IP network.

Appendix

84. The control device of claim 83, the further improvement wherein the control device registers the characteristic with in a Javaspace on the IP network.

85. The control device of claim 80, the further improvement wherein the control device communicates with another device over the IP network in order to obtain configuration information.

86. The control device of claim 80, the further improvement wherein the control device retains configuration information for use at startup.

87. The control device of any of claims 81 - 86, the further improvement wherein the control device has a processor that executes code that performs a control algorithm.

88. The control device of claim 87, wherein the processor executes code that configures the control device as a controller.

89. The control device of claim 88, the further improvement wherein the processor executes code to perform proportional integral derivative control.

90. The control device of any of claims 81 - 86, the further improvement wherein the control device includes a web server that any of (i) facilitates any of configuration, monitoring and maintenance of the control system or one or more control devices, (ii) collects process data from one or more control devices, (iii) generates source for operator displays, (iv) provides access to the control system, and (v) hosts an applications development environment.

91. The control device of claim 90, the further improvement wherein the control device comprises a configuration editor.

Appendix

92. The control device of any of claims 81 - 86, the further improvement wherein the control device comprises any of a web server, control station, operator console, personal computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

93. The control device of claim 92, the further improvement wherein the control device is adapted for use in a process control system.

94. A control system comprising one or more field devices or control devices according to any of claims 1 - 5, 8 - 20, 23 - 32, 34 - 43, 46 - 53, 61 - 64, 67 - 72, 75 - 78, 80 - 86, and 88 coupled via one or more IP networks.

95. A control system according to claim 94, wherein the IP network is powered.

96. A process control system having one or more field devices according to any of claims 8 - 12.

97. A control system comprising

a plurality of control devices coupled for communication via an IP network,

a DHCP server that furnishes IP addresses in response to requests by one or more of the control devices.

98. The control system of claim 97, wherein one or more of the control devices are field devices.

99. The control system of claim 98, adapted for process control, wherein the control devices are process control devices.

Appendix

100. The control system of any of claims 97 - 99, wherein the control devices comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

101. The control system of any of claims 97 - 99, wherein the DHCP server is solid state.

102. The process control system of claim 101, wherein the DHCP server is free of moving parts and comprises zero, one or more removable components.

103. The control system of any of claims 97 - 99, wherein the IP network is powered and wherein one or more of the control devices receive operational power from the IP network.

104. A DHCP server that is solid state and that is adapted for use on an IP network comprising one or more control devices.

105. The solid state DHCP server of claim 104, adapted for use with one or more control devices that are process control devices.

106. The solid state DHCP server of claim 104, adapted for use with one or more control devices that are field devices.

107. The solid state DHCP server of claim 106, adapted for use with field devices that include any of a transmitter or other sensor device, and a positioner or other actuator device.

108. A control system comprising

a plurality of control devices coupled for communication via an IP network,

a network enabler that is coupled to the IP network that responds to requests by the control devices to at least one of

Appendix

i) register names specified by the control devices,

ii) search for names specified by the control devices,

iii) posting to a network bulletin board events specified by the control devices,

v) removing from the network bulletin board events specified by the control devices,

vi) querying the network bulletin board for events specified by the control devices,

vii) notifying the control devices of events specified by the control devices.

109. The control system of claim 108, wherein the network enabler is any of a JINI and a JavaSpace server.

110. The control system of claim 108, wherein one or more of the control devices are field devices.

111. The control system of claim 110, adapted for process control, wherein the control devices are process control devices.

112. The control system of any of claims 108 - 111, wherein the control devices comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

113. The control system of any of claims 108 - 111, wherein the network enabler is solid state.

114. The process control system of claim 113, wherein the network enabler is free of moving parts and comprises zero, one or more removable components.

Appendix

115. The control system of any of claims 108 - 111, wherein the IP network is powered and wherein one or more of the control devices receive operational power from the IP network.

116. A network enabler comprising

at least one IP network interface, and

one or more server functionalities, each of which responds to requests received over the network interface to at least one of

i) register a name specified in a request,

ii) search for a name specified in a request,

iii) post to a network bulletin board an event specified in a request,

v) remove from the network bulletin board an event specified in a request,

vi) query the network bulletin board for an event specified in a request,

vii) notify a requestor of events specified thereby,

viii)serve as a web server.

117. The network enabler of claim 116, wherein the server functionality is any of a JINI and a JavaSpace server.

118. The network enabler of any of claims 116 - 117, wherein the network enabler is solid state.

Appendix

119. The network enabler of claim 118, wherein the network enabler is free of moving parts and comprises zero, one or more removable components.

120. The network enabler of claim 118, wherein the network enabler receives operational power from the IP network.

121. A method of operating a field device for a control system, the improvement comprising the steps of

providing within the field device a virtual machine environment,

executing byte code embodying a control algorithm within the virtual machine environment.

122. The method of claim 121, the further improvement comprising executing a control function block with the byte code.

123. The method of claim 121, the further improvement comprising configuring the field device as a controller.

124. The method of claim 123, the further improvement comprising configuring the field device with the byte code to perform proportional integral derivative control.

125. A method of operating a field device for a control system, the improvement comprising the steps of the steps of

providing a virtual machine environment within the field device ,

executing byte code within the virtual machine environment to configure the field device to perform signal conditioning.

Appendix

126. The method of any of claims 121 - 125, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

127. The method of any of claims 121 - 125, the further improvement wherein the byte code comprises JAVA byte code.

128. A method of operating a field device for process control, the improvement comprising executing a web server within the field device.

129. The method of claim 128, the further improvement wherein the web server is embedded.

130. The method of claim 128, the further improvement comprising serving with the web server web pages that facilitate any of configuration, monitoring and maintenance of at least the field device.

131. The method of claim 130, the further improvement comprising controlling configuration of at least the field device with a configuration editor that utilizes the web server as an interface.

132. The method of claim 131, the further improvement comprising selectively disabling or enabling the configuration editor, depending upon the type of network to which the field device is coupled.

133. A method of operating a field device for a control system, the improvement comprising the steps of executing a web server and a virtual machine environment within the field device.

134. The method of claim 133, the further improvement comprising executing byte code within the virtual machine environment that configures the field device to any of (i) execute a control algorithm and (ii) perform signal conditioning.

Appendix

135. The method of claim 134, the further improvement comprising configuring the field device as a controller.

136. The method of claim 135, the further improvement comprising configuring the field device with the byte code to perform proportional integral derivative control.

137. The method of claim 134, the further improvement wherein the web server is embedded.

138. The method of claim 134, the further improvement comprising facilitating any of configuration, monitoring and maintenance of at least the field device with the web server.

139. The method of claim 138, the further improvement comprising controlling at least the field devices' configuration with a configuration editor that utilizes the web server as an interface.

140. The method of claim 139, the further improvement comprising selectively disabling or enabling the configuration editor, depending upon the type of network to which the field device is coupled.

141. The method of any of claims 133 - 140, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

142. The method of any of claims 133 - 140, the further improvement wherein the byte code comprises JAVA byte code.

143. A method of operating a field device for a control system, the improvement comprising the steps of interfacing the field device with other elements of the control system via IP network.

Appendix

144. The method of claim 143, the further improvement comprising receiving operational power for the field device from the IP network.

145. The method of claim 143, the further improvement comprising interfacing the field device with other elements of the control system via an Ethernet network.

146. The method of claim 145, the further improvement comprising receiving operational power for the field device from the Ethernet network.

147. The method of claim 143, the further improvement comprising providing a processor in the field device that is in communication with the IP network.

148. The method of claim 147, the further improvement wherein the processor is a low-power processor.

149. The method of claim 147, the further improvement comprising executing a virtual machine environment and a web server with the processor.

150. The method of claim 149, the further improvement comprising executing an operating system with the processor..

151. The method of claim 150, the further improvement comprising executing a real-time operating system with the processor.

152. The method of claim 150, the further improvement wherein the field device comprises at least one of a random access memory, a read-only memory, FlashRAM, and a sensor interface.

Appendix

153. The method of any of claims 143 - 152, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

154. A method of operating a field device for process control, the improvement comprising the steps of

interfacing the field device with other elements of the control system via an IP network,

operating a processor in the field device that is in communication with the IP network.

155. The method of claim 154, the further improvement comprising receiving operational power for the field device from the IP network.

156. The method of claim 154, the further improvement comprising interfacing the field device with other elements of the control system via an Ethernet network.

157. The method of claim 156, the further improvement comprising receiving operational power for the field device from the Ethernet network.

158. The method of claim 154, the further improvement comprising executing a web server with the processor.

159. The method of claim 154, the further improvement comprising operating the processor to execute any of (i) a control algorithm and (ii) a signal conditioning algorithm.

160. The method of claim 159, the further improvement comprising performing proportional integral derivative control with the processor.

Appendix

161. The method of claim 154, the further improvement comprising executing with the processor a web server that facilitates any of configuration, monitoring and maintenance of at least the field device.

162. The method of claim 161, the further improvement comprising controlling configuration of at least the field device with a configuration editor that utilizes the web server as an interface.

163. The method of claim 162, the further improvement comprising selectively disabling or enabling the configuration editor, depending upon the type of network to which the field device is coupled.

164. The method of any of claims 154 - 163, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

165. The method of claim 164, the further improvement comprising utilizing the field processor for process control.

166. A method of operating a field device for a control system, the improvement comprising the steps of

interfacing the field device with other elements of the control system via an IP network,

issuing a request from the field device to the IP network for an IP address.

167. The method of claim 166, the further improvement comprising receiving a device identification name from any of (i) a user-configured hub or other device to which the field device is coupled, (ii) a letterbug installed in the field device, (iii) a digital data processing apparatus, (iv) a software generated letterbug.

Appendix

168. The method of claim 166, the further improvement comprising registering a characteristic of the field device via the IP network.

169. The method of claim 168, the further improvement comprising registering the characteristic with a bulletin board on the IP network.

170. The method of claim 169, the further improvement comprising registering the characteristic with in a Javaspace on the IP network.

171. The method of claim 166, the further improvement comprising communicating between the field and another element of the system over the IP network in order to obtain configuration for the field device.

172. The method of claim 166, the further improvement comprising retaining configuration information within the field device for use at startup.

173. The method of any of claims 166 - 172, the further improvement comprising performing with a processor in the field device any of (i) a control algorithm and (ii) signal conditioning.

174. The method of claim 173, the further improvement comprising configuring the field device as a controller.

175. The method of claim 174, the further improvement comprising configuring the field device to perform proportional integral derivative control.

176. The method of any of claims 166 - 172, the further improvement comprising executing a web server within the field device, the web server facilitating any of configuration, monitoring and maintenance of at least the field device.

Appendix

177. The method of claim 176, the further improvement comprising executing a configuration editor within the field device and providing an interface to the configuration editor via the web server.

178. The method of claim 177, the further improvement comprising selectively disabling or enabling the configuration editor, depending upon the type of network to which the field device is coupled.

179. The method of any of claims 166 - 172, the further improvement wherein the field device comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

180. The method of claim 179, the further improvement comprising utilizing the field processor for process control.

181. A method of operating a control device for a control system, the improvement comprising the steps of

executing byte code within a virtual machine environment provided in the control device,

the byte code configuring the control device to execute a control algorithm.

182. The method of claim 181, the further improvement comprising using the byte code to configure the control device to execute a control function block.

183. The method of claim 181, the further improvement comprising using the byte code to configure the control device as a controller.

184. The method of claim 183, the further improvement comprising using the byte code to configure the control device to perform proportional integral derivative control.

Appendix

185. The control device of any of claims 181 - 184, the further improvement wherein the control device comprises any of web server, control station, operator console, personal computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

186. The control device of any of claims 181 - 184, the further improvement wherein the byte code comprises JAVA byte code.

187. A method of operating a control device for a control system, the improvement comprising executing a web server and a virtual machine environment within the control device.

188. The method of claim 187, the further improvement comprising using the byte code to configure the control device to execute a control algorithm.

189. The method of claim 188, the further improvement comprising using the byte code to configure the control device as a controller.

190. The method of claim 189, the further improvement comprising using the byte code to configure the control device to perform proportional integral derivative control.

191. The method of claim 187, the further improvement wherein any of the web server and the virtual machine environment is embedded.

192. The method of claim 187, the further improvement comprising using the web server to any of (i) facilitate any of configuration, monitoring and maintenance of the control system or one or more control devices, (ii) collect process data from one or more control devices, (iii) generate source for operator displays, (iv) provide access to the control system, and (v) host an applications development environment.

193. The control device of any of claims 187 - 192, the further improvement wherein the control device comprises any of a web server, control station, operator console, personal

Appendix

computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

194. The control device of any of claims 187 - 192, the further improvement wherein the byte code comprises JAVA byte code.

195. A method of operating a control device for a control system, the improvement comprising the steps of

operating a low-power processor within the control device,

communicating between the control device and other elements of the control system via an IP network,

drawing operational power for the control device from the IP network.

196. The method of claim 195, the further improvement comprising interfacing the field device with other elements of the control system via an Ethernet network.

197. The method of claim 195, the further improvement comprising executing a virtual machine environment and a web server with the processor.

198. The method of claim 197, the further improvement wherein the control device comprises at least one of a random access memory, a read-only memory, FlashRAM, and a sensor interface, access to permanent storage, a configurator, system management software, messaging services, alarm/event notification, byte code implementing process control functions, byte code implementing status reporting.

199. The control device of any of claims 195 - 198, the further improvement wherein the control device comprises any of a web server, control station, operator console, personal

Appendix

computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

200. A method of operating a control device for process control, the improvement comprising the steps of

communicating between the control device and other elements of the control system via an IP network, and

causing the control device, on startup, to register a characteristic thereof with at least one other device over the IP network.

201. The method of claim 200, the further improvement comprising issuing with the control device, via the IP network, a request for assignment of an IP address.

202. The method of claim 200, the further improvement comprising receiving for the control device a device identification name from any of (i) a user-configured hub or other device to which the control device is coupled, (ii) a letterbug installed in the control device, (iii) a digital data processing apparatus, (iv) a software generated letterbug.

203. The method of claim 200, the further improvement comprising registering a characteristic of the control device with a bulletin board on the IP network.

204. The method of claim 203, the further improvement comprising registering the characteristic with in a Javaspace on the IP network.

205. The method of claim 200, the further improvement comprising communicating between the control device and another device over the IP network in order to obtain configuration information for the control device.

Appendix

206. The method of claim 200, the further improvement comprising retaining configuration information within the control device for use at startup.

207. The control device of any of claims 201 - 206, the further improvement comprising executing code embodying a control algorithm on a processor within the control device.

208. The method of claim 207, comprising using the code to configure the control device as a controller.

209. The method of claim 208, the further improvement wherein the processor executes code to perform proportional integral derivative control.

210. The control device of any of claims 201 - 206, the further improvement comprising executing a web server within the control device, the web server any of (i) facilitating any of configuration, monitoring and maintenance of the control system or one or more control devices, (ii) collecting process data from one or more control devices, (iii) generating source for operator displays, (iv) provides access to the control system, and (v) hosting an applications development environment.

211. The method of claim 210, the further improvement comprising executing a configuration editor within the control device, the configuration editor utilizing the web server as an interface.

212. The control device of any of claims 201 - 206, the further improvement wherein the control device comprises any of a web server, control station, operator console, personal computer, handheld computer, workstation, integrator, controller, transmitter or other sensor device, positioner or other actuator device.

213. The method of claim 212, the further improvement comprising utilizing the control device for process control.

Appendix

214. A method of operating control system comprising the steps of

communicating among a plurality of control devices via an IP network,

generating, within at least a selected one of the control devices, a request for an IP address,

furnishing IP addresses to a requesting control device with a DHCP server in response to such a request.

215. The method of operating a control system of claim 214, wherein one or more of the control devices are field devices.

216. The method of operating a control system of claim 215, including the step of utilizing the control devices for process control.

217. The method of operating a control system of any of claims 214 - 216, wherein the control devices comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

218. The method of operating a control system of any of claims 214 - 216, comprising the step of providing a DHCP server that is solid state.

219. The process control system of claim 218, comprising providing a DHCP server that is free of moving parts and that comprises zero, one or more removable components.

220. The method of operating a control system of any of claims 214 - 216, comprising the step of drawing operational power for the control devices from the IP network.

221. A method of operating a control system comprising the steps of

Appendix

communicating among a plurality of control devices coupled via an IP network,

responding to requests issued by the control devices over the IP network with a network enabler that at least one of

i) registers names specified by the control devices,

ii) searches for names specified by the control devices,

iii) posts to a network bulletin board events specified by the control devices,

v) removes from the network bulletin board events specified by the control devices,

vi) queries the network bulletin board for events specified by the control devices,

vii) notifies the control devices of events specified by the control devices.

222. The method of operating a control system of claim 221, wherein the network enabler is any of a JINI and a JavaSpace server.

223. The method of operating a control system of claim 221, wherein one or more of the control devices are field devices.

224. The method of operating a control system of claim 223, for process control.

225. The method of operating a control system of any of claims 221 - 224, wherein the control devices comprises any of a transmitter or other sensor device, and a positioner or other actuator device.

Appendix

226. The method of operating a control system of any of claims 221 - 225, comprising the step of providing a network enabler that is solid state.

227. The process control system of claim 226, comprising providing a network enabler that is free of moving parts and comprises zero, one or more removable components.

228. The method of operating a control system of any of claims 221 - 224, comprising the step of drawing operational power for the control devices from the IP network.

Appendix

Abstract of the Invention

The invention provides improved methods and apparatus for control using field and control devices that provide a virtual machine environment and that communicate via an IP network. By way of non-limiting example, such field device can be an "intelligent" transmitter or actuator that includes a low power processor, along with a random access memory, a read-only memory, FlashRAM, and a sensor interface. The processor can execute a real-time operating system, as well as a Java virtual machine (JVM). Java byte code executes in the JVM to configure the field device to perform typical process control functions, e.g., for proportional integral derivative (PID) control and signal conditioning. Control networks can include a plurality of such field and control devices interconnected by an IP network, such as an Ethernet.

Appendix (from PCT Patent Application US00/10013, filed April 14, 2000) Appendix

Appendix to Patent Application for

5

METHODS AND APPARATUS FOR CONTROL USING CONTROL DEVICES THAT PROVIDE A VIRTUAL MACHINE ENVIRONMENT AND THAT COMMUNICATE VIA AN IP NETWORK

Appendix (from PCT Patent Application US00/10013, filed April 14, 2000) Appendix

POWERED ETHERNET FOR INSTRUMENTATION AND CONTROL

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims priority from U.S. Provisional Application Nos. 60/129,541 and 60/157,110, which were filed, respectively, on April 6, 1999, and October 4, 1999, both of which are incorporated by reference.

TECHNICAL FIELD

The invention relates to interconnecting devices for instrumentation and control.

BACKGROUND

Control systems often include intelligent field devices having digital processors. Examples of such devices include valve controllers and transmitters, such as those associated with, for example, temperature sensors, pressure sensors, and flow meters. Another type of intelligent field device is a field mounted process controller that sends control signals to, for example, a valve controller, based on process information received from one or more transmitters.

The functionality of different field devices may be incorporated into a single module. For example, a process controller, a valve controller, and a temperature sensor/transmitter may be incorporated into a single module. However, even with such consolidation of function, control systems typically require some mechanism for transmitting signals between devices, and for transmitting signals from devices to a central control station.

Due to cost constraints, connections between devices, and between the devices and a control station, are often provided using a single pair of wires. In many instances, this same pair of wires is used to provide power to the devices.

Many proprietary protocols have been developed for powering field devices and transmitting signals between the devices using a single pair of wires. Two such protocols are the HART protocol and the FOXCOMM protocol.

Appendix (from PCT Patent Application US00/10013, filed April 14, 2000) Appendix

The HART protocol specifies provision of a 4-20 mA analog control signal and a 1200 baud bidirectional digital data link on a pair of wires connected to a device. The 4-20 mA signal is used both to power the device and to provide a control signal to or from the device. The HART protocol achieves simultaneous analog and digital transmission by using a frequency shift keying (FSK) technique to overlay a bidirectional digital signal on the analog control signal.

The FOXCOMM protocol also overlays a bidirectional digital data signal on a 4-20 mA analog control and power signal. Different versions of the protocol use either 600 baud or 4800 baud digital signals. Yet another version uses the analog signal only for power delivery and transmits all information using digital signals.

SUMMARY

In one general aspect, the invention features providing a powered Ethernet connection between two devices connected by an Ethernet connection that provides data communication between the devices. Electrical power is applied on to the Ethernet connection at the first device. At the second device, the electrical power is extracted and used to power the second device.

Embodiments may include one or more of the following features. For example, the Ethernet connection may include two pairs of wires, with a first pair of wires being used to transmit data from the first device to the second device and a second pair of wires being used to transmit data from the second device to the first device. In this case, applying electrical power on to the Ethernet connection at the first device may include applying a DC voltage across the two pairs of wires by coupling a first potential to the first pair of wires and a second potential to the second pair of wires, with the DC voltage being defined as a difference between the two potentials. For example, when each pair of wires is connected to a corresponding transformer at the first device, and each transformer includes a center-tapped primary winding, coupling potentials to the pairs of wires may include applying the DC voltage between the center taps of the primary windings of the two transformers. As an alternative, a center-tapped inductor may be connected across each pair of wires at the first device, and potentials may be coupled to the pairs of wires by applying the DC voltage between the center taps of the inductors. Electrical power may be extracted at the second

device through center-tapped windings of transformers of the second device, or through center-tapped inductors coupled across the pairs of wires at the second device.

The Ethernet connection also may be implemented using a single pair of wires to transmit data from the first device to the second device, to transmit data from the second device to the first device, and to provide power from the first device to the second device. In this case, applying electrical power on to the Ethernet connection at the first device may include applying a DC voltage across the pair of wires by coupling a first potential to the first wire through an inductor and coupling a second potential to the second wire through an inductor, with the DC voltage being defined as a difference between the two potentials.

Each device may be, for example, a process controller, a field device, or an Ethernet hub. A field device may be operable to sense a process condition and to transmit information about the sensed process condition using the Ethernet connection, or to control a process condition in response to a command received through the Ethernet connection.

In another general aspect, the invention features connecting a four-terminal Ethernet connection to a two-wire Ethernet connection. A device having a four-terminal Ethernet connection, the connection including a first pair of terminals for transmitting data away from the device and a second pair of terminals for transmitting data to the device is connected to a two-wire Ethernet connection for transmitting data to and from the device by a switched connection. The switched connection operating in a first mode in which the two-wire Ethernet connection is connected to the first pair of terminals and a second mode in which the two-wire Ethernet connection is connected to the second pair of terminals. The switched connection is initially set to operate in the first mode, and the two-wire Ethernet connection is monitored for data being transmitted to the device. The switched connection is set to operate in the second mode upon detection of data being transmitted to the device.

Power may be injected on to the two-wire Ethernet connection at the device for use in powering another device connected to the two-wire Ethernet connection. In addition, an impedance presented to the two-wire Ethernet connection may be changed from a first impedance when the switched connection is operating in the first mode to a second impedance when the switched connection is operating in the second mode.

In another general aspect, the invention features a field device having a powered Ethernet connection. The field device includes communications circuitry operable to

communicate data over an Ethernet connection using an Ethernet protocol, power circuitry operable to extract operating power from the Ethernet connection, and process circuitry operable to sense a process condition and to transmit information about the sensed process condition using the Ethernet connection, or to control a process condition in response to a command received through the Ethernet connection.

Examples of such a field device include a device providing a temperature or pressure sensor, a flow meter, and a valve controller. The communications circuitry and the power circuitry may be operable to interface with an Ethernet connection including only a single pair of wires.

Other features and advantages will be apparent from the following description, including the drawings, and from the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Figs. 1-3 are block diagrams of systems using powered Ethernet connections.

Figs. 4 and 5 are block diagrams of systems for providing power to a field device over a four-wire Ethernet connection.

Fig. 6 is a circuit diagram of a four-wire circuit.

Fig. 7 is a block diagram of a system for providing power and emulating a four-wire Ethernet connection using only a single pair of wires.

Figs. 8A-8F are block diagrams of different operating states of the system of Fig. 7.

Fig. 9 is a circuit diagram showing connections of a hub using two-wire powered Ethernet.

Fig. 10 is a circuit diagram of a line circuit of the circuit of Fig. 9.

Figs. 11-17 are circuit diagrams of elements of a two-wire circuit.

DETAILED DESCRIPTION

Referring to Fig. 1, a control system 100 includes a controller 105 that powers and controls a remote field device 110 over a powered Ethernet connection 115. Thus, the system 100 applies the well-known and well-established Ethernet protocol to an instrumentation and control system.

Appendix (from PCT Patent Application US00/10013, filed April 14, 2000) Appendix

The powered Ethernet connection 115 may be provided by modifying connections to existing wiring of a system employing a proprietary protocol. Thus, a powered Ethernet system may be produced by retrofitting an existing system.

Protocols such as the Ethernet protocol are often used to communicate between devices, such as components of a computer network. Ethernet uses a bus or star topology and supports data transfer rates of 10 Mbps. Ethernet handles simultaneous demands using a technique known as carrier sense multi-access with collision detection ("CSMA/CD"). The Ethernet specification served as the basis for the IEEE 802.3 communication standard, which specifies the physical and lower software layers. The IEEE 802.3 standard controls the implementation of the basic Ethernet hardware technology. In specific terms, it controls the media access control sublayer and the physical layer functions for a bus-structured local area network that uses CSMA/CD as an access protocol.

One of several adaptations of the Ethernet (IEEE 802.3) standard for Local Area Networks (LANs) is 10Base-T. The 10Base-T standard (also called Twisted Pair Ethernet) uses a twisted-pair cable with maximum lengths of 100 meters. The cable is thinner and more flexible than the coaxial cable used for the 10Base-2 or 10Base-5 standards. Cables in the 10Base-T system connect with RJ-45 connectors. A star topology is common, with 12 or more computers connected directly to a hub or a concentrator. The 10Base-T system operates at 10 Mbps and uses baseband transmission methods.

A networking standard that supports data transfer rates up to 100 Mbps (100 megabits per second), called 100BASE-T, is based on the older Ethernet standard. Because it is 10 times faster than Ethernet, it is often referred to as Fast Ethernet. Officially, the 100BASE-T standard is IEEE 802.3u.

Information is communicated on the Ethernet using electrical signals. The standard Ethernet protocol specifies the physical layer characteristics of these signals. This information is converted to digital information by the Ethernet devices.

The Ethernet protocol typically employs a first pair of wires for transmission in one direction and a second pair of wires for transmission in the opposite direction. The pairs of wires generally are not used to power the devices. Associated with the Ethernet protocol is a well-developed collection of standard hardware and software components that may be used to implement communications between devices.

Appendix (from PCT Patent Application US00/10013, filed April 14, 2000) Appendix

The powered Ethernet connection 115 provides a connection for powering and communicating with remote field devices using standard, or slightly modified, Ethernet hardware and software. The connection 115 may employ either one or two pairs of wires.

Figs. 2 and 3 illustrate examples of other system topologies. As shown in Fig. 2, a system 200 includes a controller 205 connected to a collection of field devices 210 by a powered Ethernet connection 215 in a bus configuration. The connection 215 may employ either one or two pairs of wires.

Similarly, Fig. 3 shows a system 300 that includes a controller 305 connected to a hub 310 by a powered Ethernet connection 315. The hub 310 is then connected to a collection of field devices 320 by powered Ethernet connections 325 in a star configuration.

Each of the connections 315, 325 may employ either one or two pairs of wires, and different connections may employ different numbers of pairs. For example, the connection 315, which is typically substantially longer than the connections 325, may employ a single pair of wires, while one or more of the connections 325 employs two pairs of wires. Exemplary single-pair (two-wire) and double-pair (four-wire) systems are discussed in detail below.

Four-Wire System

Referring to Fig. 4, a system 400 employs standard Ethernet boards 405 at both a process controller 410 and a remote field device 415. The boards communicate via two pairs of wires, with the first pair 420 providing communications from the process controller to the field device, and the second pair 425 providing communications from the field device to the process controller.

Each Ethernet board includes a pair of transformers 430, with each transformer being associated with a pair of wires. At the process controller, a 24 Volt DC power signal from a power supply 435 is injected to central taps 440 of the primary coils of the transformers. The positive (+V) lead 445 of the DC power signal is provided at the central tap of the primary coil of one of the transformers. The ground (GND) lead 450 of the DC power signal is provided at the central tap of the primary coil of the other transformer.

At the field device, the central tap 440 of the primary coil of one of the transformers 430 serves as device ground 455. The central tap 440 of the primary coil of the other

transformer 430 provides a device operating power signal 460. The signal 460 and ground 455 are provided to a DC-to-DC converter 465 that reduces and isolates the DC voltage to provide operating power for the device. For optimum power transfer, the coil of the DC-to-DC converter can be impedance matched to the resistance of the wire pairs. The output of the DC-to-DC converter is provided to a voltage regulator circuit 470 that provides power to the Ethernet board and the field device. The process controller also may include a DC-to-DC converter 465 and a voltage regulator circuit 470 for use in powering the Ethernet board of the process controller. These elements are unnecessary when the process controller has a separate power supply.

Only one remote field device 415 is shown in Fig. 4. However, as illustrated in Figs. 2 and 3, nothing precludes having more devices connected at the far end of the Ethernet wire pairs. The only requirements are that the loads of the devices must be designed so that the wire pairs are properly terminated and that the total power delivered is sufficient to drive all of the devices. Since only one remote device is active at a time, the power requirement will be the sum of the requirements of the one active device and N - 1 idle devices, where N is the number of devices connected to the wire pair.

The transformers include primary coils connected to the twisted pairs and secondary coils connected to devices on the Ethernet boards. The secondary coils of the transformers may be constructed of bi-filar coils to reduce the required size of the transformers.

Referring to Fig. 5, for transformers that cannot carry the required DC current or don't have center taps in their primary coils, the power can be applied via center-tapped inductors 500 connected across the secondary coils of the transformers. The DC current is isolated from the transformers by capacitors 505. Each inductor must be large enough that it does not degrade the termination impedance of its corresponding wire pair.

Fig. 6 illustrates a fairly simple prototype circuit 600 for powering a four-wire connection 605. At the process controller end 610, center-tapped inductors 615 are used to direct power on to the connection 605. At the field device end 620, center-tapped inductors 625 are used to remove power from the connection 605. With the circuit shown in Fig. 6, a 24 volt signal was applied to the connection 605 at the process controller side and about 100 mA was drained through a resistor 630 on the field device end, resulting in power consumption of 2.5 watts. The two ends were implemented using NIC cards 635 mounted in

personal computers. No attempt was made to actually power any part of the NIC cards. The loop was about six feet long, which is much shorter than many contemplated implementations. The prototype circuit confirmed that power can be successfully injected into an Ethernet connection without jeopardizing the communications capability of the connection.

Two-Wire System

Referring to Fig. 7, a system 700 having a two-wire connection may be used to emulate a four-wire, powered Ethernet connection. This approach is particularly useful in retrofitting Ethernet to existing systems that use only a single pair of wires to provide power and communications between a process controller and a remote field device. In particular, this approach may be used to make Ethernet connections without upgrading or replacing existing wires.

The system 700 includes a hub port 705 connected by a single pair of wires 710 to a remote field device 715. Each connection includes the hub port 705, a switch 720 connected between the port and the hub end of the single pair of wires 710, an activity detector 725 connected to the switch 720, and a switch 730 connected between the other end of the pair of wires and the remote field device. Each switch functions as a double pole, double throw switch, and may be implemented with a driver and receiver that can be enabled or disabled by means of a chip select lead. The switches control their associated devices to be in reception states or transmission states.

Referring to Fig. 8A, the remote field device 715 and an associated hub port 705 operate in an idle mode when no traffic is present on the wire connection between them. The hub switch 720 is switched to the transmit position and the remote field switch 730 is switched to the receive position. In this mode, the remote field device can determine if the network is clear.

Referring to Fig. 8B, when the field device 715 is ready to send data, the field device 715 switches the switch 730 to the transmit position under control of the Ethernet remote physical layer. This connects the field device to the hub activity detector 725. The activity detector 725 detects the first part of the output signal of the remote device 715.

Referring to Fig. 8C, once the hub's activity detector 725 detects activity, it switches the switch 720 to the receive position to connect the transmitting field device to the transmit pair of the hub. There may be several bits of transmission lost during this transition. However, this causes no problem since the first 64 bits of the Ethernet packet are only used for synchronizing clocks.

Referring to Fig. 8D, once the hub starts receiving data, the hub generates a signal to the other remote field devices to notify them that the system is busy so that they will not start to send data of their own.

Referring to Fig. 8E, the hub may detect a collision when it receives a signal on its receive pair from another remote field device connected to it. It must then signal to the remote field devices that a collision has occurred. This can be done in several ways. One simple way is to reverse the polarity of the DC power supplied to the field devices. The field devices may be configured to automatically switch their switches to the receive positions in response to this polarity reversal, as shown in Fig. 8F. The transmitting field devices will then see the jamming signal on their receive inputs, will stop transmitting, and will initiate a backout sequence to resolve the collision.

Hub

Referring to Fig. 9, an eight-port, standard Ethernet hub 900 includes ports 905, each of which includes a two-wire transmit connection 910 that receives data transmitted by a remote device 715 and a two-wire receive connection 915 that sends data to be received by the remote device 715. A two-wire powered Ethernet connection is provided by inserting a line circuit 920 between a hub port 905 and a two-wire connection 710 to a remote device 715. This permits the use of standard Ethernet hubs to provide powered connections to remote devices over only a single pair of wires.

Referring to Fig. 10, each hub port line circuit 920 includes a first pair of terminals 1000 connected to the two-wire connection 710, a second pair of terminals 1005 connected to the transmit connection 910, and a third pair of terminals 1010 connected to the receive connection 915. Power is supplied to the terminals 1000 by a 24 Volt source connected to the terminals through inductors 1015 or other high impedance devices. Capacitors 1020

serve to isolate the power from a transformer 1025, the secondary coil of which is connected through the capacitors 1020 to the terminals 1000.

A load switch 1030 is connected across the primary coil of the transformer 1025. The load switch 1030 is controlled by the hub to provide impedance matching. When the hub port is in the receive mode, the load switch 1030 connects a load impedance across the line so that the line is properly terminated. When the hub port is in the transmit mode, the load switch removes the load since a transmission driver 1035 provides the terminating impedance.

An activity detector 1040 also is connected across the primary coil of the transformer 1025. As discussed above, the activity detector 1040 detects transmissions by the remote device and notifies the hub to permit the hub to reconfigure the line circuit 920. The activity detector can be implemented in a number of ways. One simple implementation rectifies the signal produced by the primary coil of the transformer 1025 and compares the rectified signal to a threshold that indicates that a signal is present.

A receiver 1045 is connected across the primary coil of the transformer 1025. Outputs of the receiver 1045 are connected to the pair of terminals 1005. The receiver 1045 is controlled by a signal (TCE) from the hub.

Outputs of the transmission driver 1035 are connected through resistors 1050 to the primary coil of the transformer 1025. The resistors 1050 may be used to provide a 100 ohm driving impedance. Inputs of the transmission driver 1035 are connected to the pair of terminals 1010. The transmission driver 1035 is controlled by a signal (RCE) from the hub.

Two-Wire System - Details

As discussed above with reference to Fig. 7, a powered, two-wire 10 MHz half duplex Ethernet communications system may be used to transmit Ethernet packets as well as DC power to remote field devices. This permits existing 4-20 mA loop-powered remote field devices to be upgraded to intelligent field devices using their existing 4-20 mA twisted pair current loop cabling. This eliminates the substantial expense of replacing existing current loop wiring to upgrade the customer's cable to UTP Category 5 Ethernet wire.

To understand how the two-wire powered Ethernet communication system operates, it is useful to understand how conventional 10 BaseT Ethernet operates. To this end, three

conventional Ethernet configurations are considered below: (1) point-to-point communication, (2) multi-point communication through an Ethernet hub, and (3) multi-point communication through an Ethernet switch.

Point-to-Point

The first configuration examined was a two-wire powered Ethernet point-to-point connection that involved two NIC cards connected together via a cable that was cut in half so that the two pairs of the cable could be connected together. With the two pairs shorted together, the NIC cards were put in the full-duplex mode so that a card that received its own signal would not interpret the signal as a collision and shut down. The two cards communicated with each other, by "pinging" each other continuously. Pinging is simply sending a message with an address of a device. If the device is up and connected, it sends an acknowledgement.

This configuration used two standard NIC cards from SMC installed on two standard personal computers running Windows NT. The PHY chip on the SMC NIC cards is the Altima AH101 single port chip. As noted above, the TX output of each NIC card was connected to its RX input to reduce the channel from four wires down to two. Both cards were set to 10MHz, full-duplex operation in Windows NT using the NIC cards' standard driver software. This setting did not allow the two NIC cards to perform auto-negotiation, which would have caused communications to fail. In addition, the cable was a short run (approximately 10 ft.) of Category 5 unshielded twisted pair.

No hardware changes were made to the NIC cards themselves. The two wire connection of the TX pair to the RX pair was made external to the NIC card on a small adapter board. In addition, no power was sent during initial testing.

DC power was later introduced into this configuration using several discrete analog components. These modifications were added to the external adapter board. Additions made to the adapter boards to superimpose DC power into the two-wire cable included using inductors to couple DC power on to the two-wire composite pair, while adding capacitors to AC couple the high speed Ethernet signals to the wires. Ten volts of DC power was applied to one adapter board and was sent across a short run of Category 5 unshielded twisted pair to the other adapter board on which was mounted a 10 Ohm, 10 Watt power resistor. With 10

volts sent across the two-wire composite pair, a current of 100 mA was delivered to the resistor, which resulted in a total of 1 Watt being sent across the cable, simultaneous with Ethernet signal information (i.e., pinging) being sent between the two NIC cards.

Multi-Point with Hub (No Modifications to Hub)

The second configuration, which proved unsuccessful, employed a standard NIC card connected to a standard hub and back to a standard NIC card. The configuration included one NIC card, again the SMC 9432 NIC card using the Altima AH101 PHY chip, connected in two-wire mode via UTP Category 5 cable to one port of a standard low cost single board Ethernet hub. Another 9432 NIC card (in another computer) was interfaced in standard Ethernet 4 wire mode via a short run (10 ft.) of UTP Category 5 cabling to another port of the Ethernet hub. When tested, this configuration failed because the hub is always a half duplex only device.

When a hub is connected in two-wire mode to some other two-wire Ethernet device, such as a NIC card, the TX output of that particular hub port may be directly connected to the RX input of that same port. Therefore, when the hub transmits out of one of its ports in two-wire mode, the TX output data is immediately received by the corresponding RX input. When a device, such as a hub, operates in half duplex mode, it can only receive when it is not transmitting and it can only transmit when it is not receiving. That is the definition of half duplex operation. Full duplex operation, on the other hand, allows a device to transmit and receive simultaneously, which is why it needs four wires. When a half duplex device receives a packet while it is transmitting, it considers this to be a collision, which is a violation of the half duplex mode of operation. The generic response of a half duplex device to a perceived collision is to immediately terminate whatever it was transmitting when something came in on its RX input, ignore the packet it is receiving on RX, generate a "jam" signal on its TX output, and then time out for some random period of time before trying to re-transmit the packet it was sending when the collision occurred. However, when it tries to re-transmit the packet, it again immediately senses the packet on its RX input, and again goes into collision response mode. This sequence of events continues indefinitely whenever the hub tries to transmit an Ethernet packet, which means that no useful information is transmitted as long as TX is connected to RX. Therefore, without hardware or firmware

modifications, a standard half-duplex, off-the-shelf hub cannot be used for two-wire Ethernet communication in a system. This problem is avoided by using a hub that functions in full duplex mode.

Multi-Point With Switch (No Hardware Modifications Made to Switch)

A third configuration used a standard NIC card interfaced to a port of a high performance Cisco switch in two-wire mode and another NIC card interfaced to the switch via another switch port in four-wire mode with no modifications made to the switch's hardware. (Changes were made to the switch firmware via the software control console on the switch.) The NIC-card-to-Cisco switch two-wire interface failed because the switch could not properly perform a needed operational function called auto-negotiation with a two-wire cable. The problems can be avoided by directing the switch to not auto-negotiate with the NIC in two-wire mode and to disregard anything it receives on its RX input when it is transmitting out of TX on its two-wire interface.

Multi-Point with Switch (Modifications Made to Switch)

A variation on the previous configuration used a low cost, single board, ADMtech switch of simple construction, made with three major VLSI components and other low cost discrete chips and components. The switch was modified by adding one low cost HC chip and a resistor to the board. The rest of the experimental configuration was the same as that for the previous configuration. In particular, one NIC card was interfaced to the switch in two-wire mode on one switch port, while a second NIC card was interfaced to the switch via another port in four-wire mode.

This configuration worked correctly and as designed because the modification to the switch prevented the switch from paying attention to packets being immediately received on the two-wire port when the TX output was transmitting out (the usual two-wire mode transmission problem). The other reason this worked is because the switch was configured during hardware setup to auto-negotiate with the NIC before it saw the two-wire interface. The lesson learned from this configuration is that a simple, low cost switch may be a viable candidate for use in a two-wire Ethernet system configuration, if the switch can be configured, through hardware or other modifications, to ignore packets being transmitted in

each two-wire port by the corresponding RX input and, through hardware or firmware modifications, to not auto-negotiate. The hardware modification to the switch involving the 74HC125 chip and resistor solves the problem of RX inputs ignoring transmitted TX packets. The second problem is solved by adding a boot strap serial EEPROM to the board to direct the main switch controller chip, the ADMtech AL968 chip, to not auto-negotiate.

The first modification of the NIC board attempted for two-wire operation is shown in Fig. 11. The NPN transistor 1100 was turned on only when TXEN went high. This connected the two 49.9 ohm resistors 1105 to TXOP and TXON. When the TXEN lead was low, the transmit driver was disconnected from the rest of the circuit. The receive pair on the output side of the transformer 1110 was connected to the transmit pair. The transmit pair was then connected to the jack 1115 and inductors were added. This approach was unsuccessful.

The next modification approach is shown in Fig. 12. All modifications were done to an external board 1200 that was then connected in series with the loop. This approach was successful and resulted in the two NIC cards communicating. One problem with this approach was that the terminations on the transmit and receive pairs were connected in parallel, which resulted in a 50 ohm termination instead of a 100 ohm termination. This is not important on short loops. With the setup shown in Fig. 12, 10 volts were applied to one card and a 100 ohm resistor on the other. This resulted in one watt being sent over the loop.

A telephone then was connected to each loop, and one end was connected to a 20-volt power supply through two 160-ohm resistors. Users were able to talk on the Ethernet loop while it was transmitting data. The circuit is shown in Fig. 13.

Other attempts to have the cards communicate through an unmodified Altima switch were unsuccessful. Since the Altima switch uses virtually the same physical layer, AHlO4L vs. AH101L, it was determined that the problem was because the switch is smarter than the NIC cards. The switch generated a table of addresses by monitoring the traffic. Once it saw its own address as the source of an incoming message, it shut down. This problem may be solved through use of a card that cuts off the receiver when the transmitter is active. Such a card is shown in Fig. 14. In Fig. 14, resistors R3 and R4 have been added to provide a DC current component since Tl is not a bipolar switch. L3 is added so that the DC current will

not saturate the transformer. With this circuit, the only modification needed on the Altima switch is to tap into the TXEN signal and to pick up a ground.

Maximum Connection Distances

The maximum distance over which powered Ethernet can operate is a function of several variables. The distance can be limited by one of two factors, the power limitation and the signal strength limitation. The power limitation is in turn determined by the power a device needs, the resistance of the loop, and the maximum voltage available for use. To get the maximum distance, the DC impedance of the converter must equal the loop resistance. In general, the connection distance provided by the four-wire mode is twice that of the two-wire mode. Assuming that the voltage is restricted to 24 volts, that 26 gauge twisted pair wiring is used, and that the power requirement for the device is one watt, the loop and the device will each dissipate one watt (since, for maximum distance, the device resistance equals the loop resistance.) The DC current will be 83.3 mA, and the loop resistance will be 144 ohms. For two-wire operation this corresponds to a loop length of 1,450 feet. For four-wire operation, the length is 2,900 ft. This is sufficiently long that the limitation due to signal strength may be the controlling factor for most applications.

While physical layer chips advertise that they can support a distance of 100 meters, Altima states that they guarantee a minimum of 140 meters at 100 Mbit operation. The limit for 10 Mbit operation should be considerably longer. Another possibility for greater signal distance is to operate at 1 Mbit. However, this would require modifications to the switch.

Full Duplex

There are a number of problems with developing true full duplex over a single pair. It requires the development of a hybrid similar to those used in the telephone. A hybrid circuit requires a point where the output signal can be picked off uncorrupted by any input signal. The present physical layer chip does not provide such a point. The chip's output is from a current source that drives the line in parallel with a 100 ohm terminating resistor. This output circuit is shown in Fig. 15. If the output were as shown in Fig. 16, the desired signal could be picked off at points a and b. One way to provide full duplex operation is shown in Fig. 17. The 100 ohm resistor connected to the two 49.9 ohm resistors is required

for the output to be the proper shape for Manchester encoding. The added circuit essentially doubles the power used by the output driver.

Multi-drop

Providing multi-drop capability is even more difficult than full duplex operation. To prevent reflections at the point of connection, the output driver must present an infinite output impedance (that is, it must be a pure current source). Since the signal injected in the loop will travel in both directions, the driver power will be doubled. The power supply also becomes more complicated. The power supplies on the remote devices must be designed so that they draw a constant current. Otherwise, near-end devices will draw excessive current and far end devices will not be able to operate.

Other embodiments are within the scope of the following claims.

What is claimed is:

Appendix (from PCT Patent Application US00/10013, filed April 14, 2000) Appendix

1. A method of providing a powered Ethernet connection between two devices, the method comprising:
providing a first device;
providing a second device;
providing an Ethernet connection between the first device and the second device, the Ethernet connection providing data communication between the devices;
applying electrical power on to the Ethernet connection at the first device;
extracting electrical power from the Ethernet connection at the second device; and
powering the second device using the extracted electrical power.

2. The method of claim 1, wherein the Ethernet connection comprises two pairs of wires, with a first pair of wires being used to transmit data from the first device to the second device and a second pair of wires being used to transmit data from the second device to the first device.

3. The method of claim 2, wherein applying electrical power on to the Ethernet connection at the first device comprises applying a DC voltage across the two pairs of wires by coupling a first potential to the first pair of wires and a second potential to the second pair of wires, with the DC voltage being defined as a difference between the two potentials.

4. The method of claim 3, wherein:
each pair of wires is connected to a corresponding transformer at the first device,
each transformer includes a center-tapped primary winding, and
coupling potentials to the pairs of wires comprises applying the DC voltage between the center taps of the primary windings of the two transformers.

5. The method of claim 3, wherein:
a center-tapped inductor is connected across each pair of wires at the first device, and
coupling potentials to the pairs of wires comprises applying the DC voltage between the center taps of the inductors.

Appendix

Appendix

6. The method of claim 3, wherein:

each pair of wires is connected to a corresponding transformer at the second device,

each transformer includes a center-tapped winding, and

extracting electrical power from the Ethernet connection at the second device comprises extracting a DC voltage existing between the center taps of the windings of the two transformers.

7. The method of claim 3, wherein:

a center-tapped inductor is connected across each pair of wires at the second device, and

extracting electrical power from the Ethernet connection at the second device comprises extracting a DC voltage existing between the center taps of the inductors.

8. The method of claim 1, wherein the Ethernet connection comprises a single pair of wires used to transmit data from the first device to the second device, to transmit data from the second device to the first device, and to provide power from the first device to the second device.

9. The method of claim 8, wherein applying electrical power on to the Ethernet connection at the first device comprises applying a DC voltage across the pair of wires by coupling a first potential to the first wire through an inductor and coupling a second potential to the second wire through an inductor, with the DC voltage being defined as a difference between the two potentials.

10. The method of claim 1, wherein the first device comprises a process controller.

11. The method of claim 10, wherein the second device comprises a field device operable to sense a process condition and to transmit information about the sensed process condition to the first device using the Ethernet connection.

12. The method of claim 10, wherein the second device comprises a field device operable to control a process condition in response to a command sent from the first device using the Ethernet connection.

13. The method of claim 1, wherein the first device comprises an Ethernet hub.

14. The method of claim 13, wherein the second device comprises a field device operable to sense a process condition and to transmit information about the sensed process condition to the first device using the Ethernet connection.

15. The method of claim 13, wherein the second device comprises a field device operable to control a process condition in response to a command sent from the first device using the Ethernet connection.

16. The method of claim 1, wherein the second device comprises a field device operable to sense a process condition and to transmit information about the sensed process condition to the first device using the Ethernet connection.

17. The method of claim 1, wherein the second device comprises a field device operable to control a process condition in response to a command sent from the first device using the Ethernet connection.

18. A method of connecting a four-terminal Ethernet connection to a two-wire Ethernet connection, the method comprising:

providing a device having a four-terminal Ethernet connection, the connection including a first pair of terminals for transmitting data away from the device and a second pair of terminals for transmitting data to the device;

providing a two-wire Ethernet connection for transmitting data to and from the device;

providing a switched connection between the four-terminal Ethernet connection and the two-wire Ethernet connection, the switched connection operating in a first mode in which

the two-wire Ethernet connection is connected to the first pair of terminals and a second mode in which the two-wire Ethernet connection is connected to the second pair of terminals;

setting the switched connection to operate in the first mode;

monitoring the two-wire Ethernet connection for data being transmitted to the device; and

setting the switched connection to operate in the second mode upon detection of data being transmitted to the device.

19. The method of claim 18, further comprising injecting power on to the two-wire Ethernet connection at the device for use in powering another device connected to the two-wire Ethernet connection.

20. The method of claim 18, further comprising changing an impedance presented to the two-wire Ethernet connection at the device from a first impedance when the switched connection is operating in the first mode to a second impedance when the switched connection is operating in the second mode.

21. A field device for use with a powered Ethernet connection, the field device comprising:

communications circuitry operable to communicate data over an Ethernet connection using an Ethernet protocol;

power circuitry operable to extract operating power from the Ethernet connection; and

process circuitry operable to:

sense a process condition and to transmit information about the sensed process condition using the Ethernet connection; or

control a process condition in response to a command received through the Ethernet connection.

22. The field device of claim 21, wherein the communications circuitry and the power circuitry are operable to interface with an Ethernet connection including only a single pair of wires.

23. A system providing a powered Ethernet connection between two devices, the system comprising:

a first device;

a second device;

an Ethernet connection between the first device and the second device, the Ethernet connection providing data communication between the devices;

circuitry for applying electrical power on to the Ethernet connection at the first device;

circuitry for extracting electrical power from the Ethernet connection at the second device; and

circuitry for powering the second device using the extracted electrical power.

24. The system of claim 23, wherein the Ethernet connection comprises two pairs of wires, with a first pair of wires being used to transmit data from the first device to the second device and a second pair of wires being used to transmit data from the second device to the first device.

25. The system of claim 24, wherein the circuitry for applying electrical power on to the Ethernet connection at the first device comprises circuitry for applying a DC voltage across the two pairs of wires by coupling a first potential to the first pair of wires and a second potential to the second pair of wires, with the DC voltage being defined as a difference between the two potentials.

26. The system of claim 25, wherein:

each pair of wires is connected to a corresponding transformer at the first device,

each transformer includes a center-tapped primary winding, and

the circuitry for coupling potentials to the pairs of wires comprises circuitry for applying the DC voltage between the center taps of the primary windings of the two transformers.

27. The system of claim 25, wherein:
a center-tapped inductor is connected across each pair of wires at the first device, and
the circuitry for coupling potentials to the pairs of wires comprises circuitry for applying the DC voltage between the center taps of the inductors.

28. The system of claim 25, wherein:
each pair of wires is connected to a corresponding transformer at the second device,
each transformer includes a center-tapped winding, and
the circuitry for extracting electrical power from the Ethernet connection at the second device comprises circuitry for extracting a DC voltage existing between the center taps of the windings of the two transformers.

29. The system of claim 25, wherein:
a center-tapped inductor is connected across each pair of wires at the second device, and
the circuitry for extracting electrical power from the Ethernet connection at the second device comprises circuitry for extracting a DC voltage existing between the center taps of the inductors.

30. The system of claim 23, wherein the Ethernet connection comprises a single pair of wires used to transmit data from the first device to the second device, to transmit data from the second device to the first device, and to provide power from the first device to the second device.

31. The system of claim 30, wherein the circuitry for applying electrical power on to the Ethernet connection at the first device comprises circuitry for applying a DC voltage across the pair of wires by coupling a first potential to the first wire through an inductor and

coupling a second potential to the second wire through an inductor, with the DC voltage being defined as a difference between the two potentials.

32. The system of claim 23, wherein the first device comprises a process controller.

33. The system of claim 32, wherein the second device comprises a field device operable to sense a process condition and to transmit information about the sensed process condition to the first device using the Ethernet connection.

34. The system of claim 32, wherein the second device comprises a field device operable to control a process condition in response to a command sent from the first device using the Ethernet connection.

35. The system of claim 23, wherein the first device comprises an Ethernet hub.

36. The system of claim 35, wherein the second device comprises a field device operable to sense a process condition and to transmit information about the sensed process condition to the first device using the Ethernet connection.

37. The system of claim 35, wherein the second device comprises a field device operable to control a process condition in response to a command sent from the first device using the Ethernet connection.

38. The system of claim 23, wherein the second device comprises a field device operable to sense a process condition and to transmit information about the sensed process condition to the first device using the Ethernet connection.

39. The system of claim 23, wherein the second device comprises a field device operable to control a process condition in response to a command sent from the first device using the Ethernet connection.

Appendix (from PCT Patent Application US00/10013, filed April 14, 2000) Appendix

ABSTRACT

A powered Ethernet connection is provided between two devices connected by an Ethernet connection that provides data communication between the devices. Electrical power is applied to the Ethernet connection at the first device, and extracted from the Ethernet connection at the second device. The extracted power is used to power the second device.

Appendix

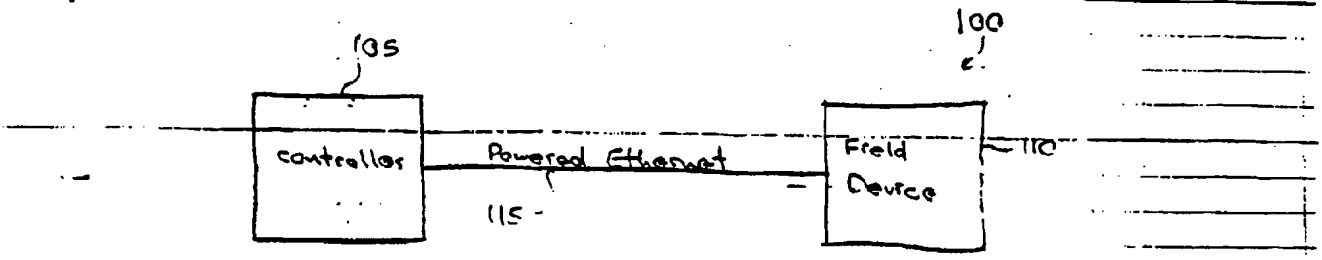


Fig. 1

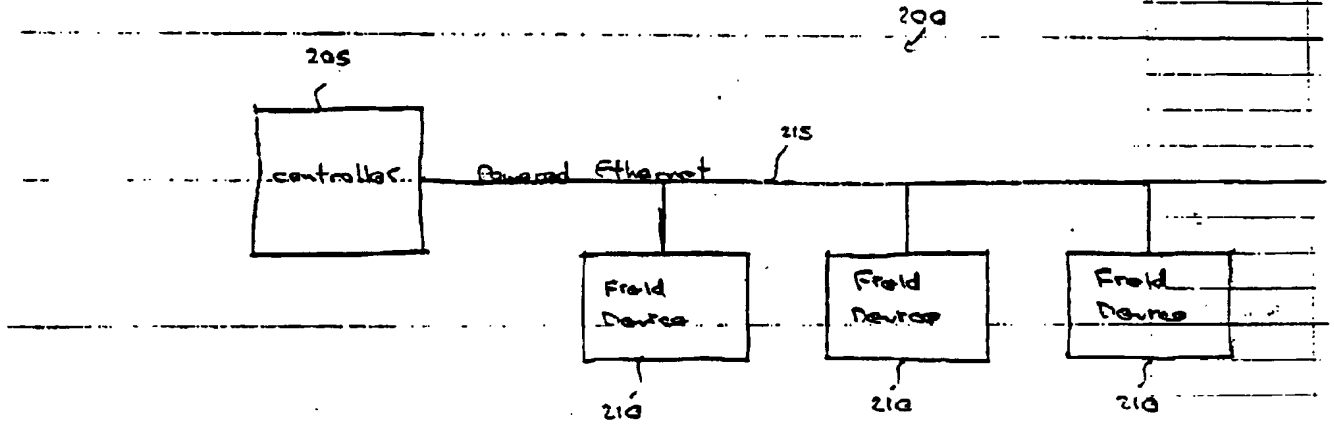


Fig. 2

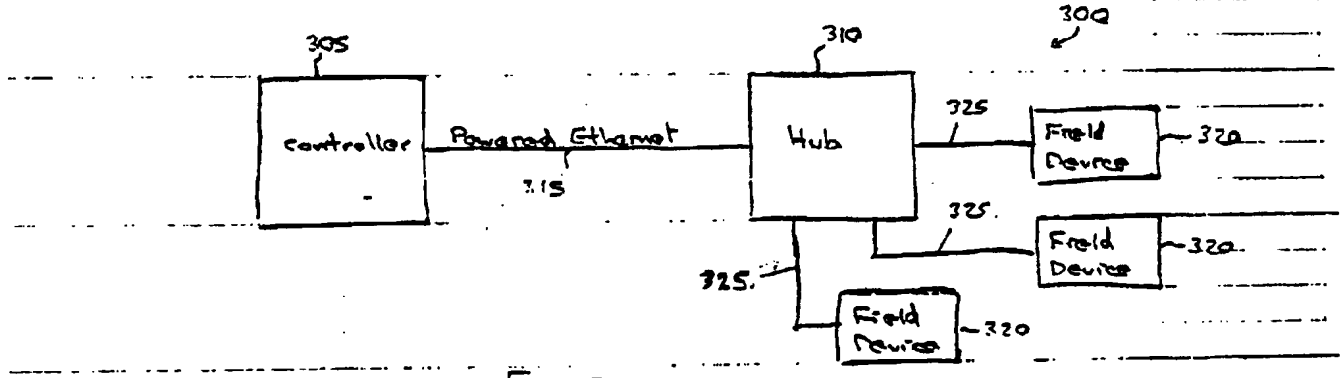


Fig. 3

Appendix

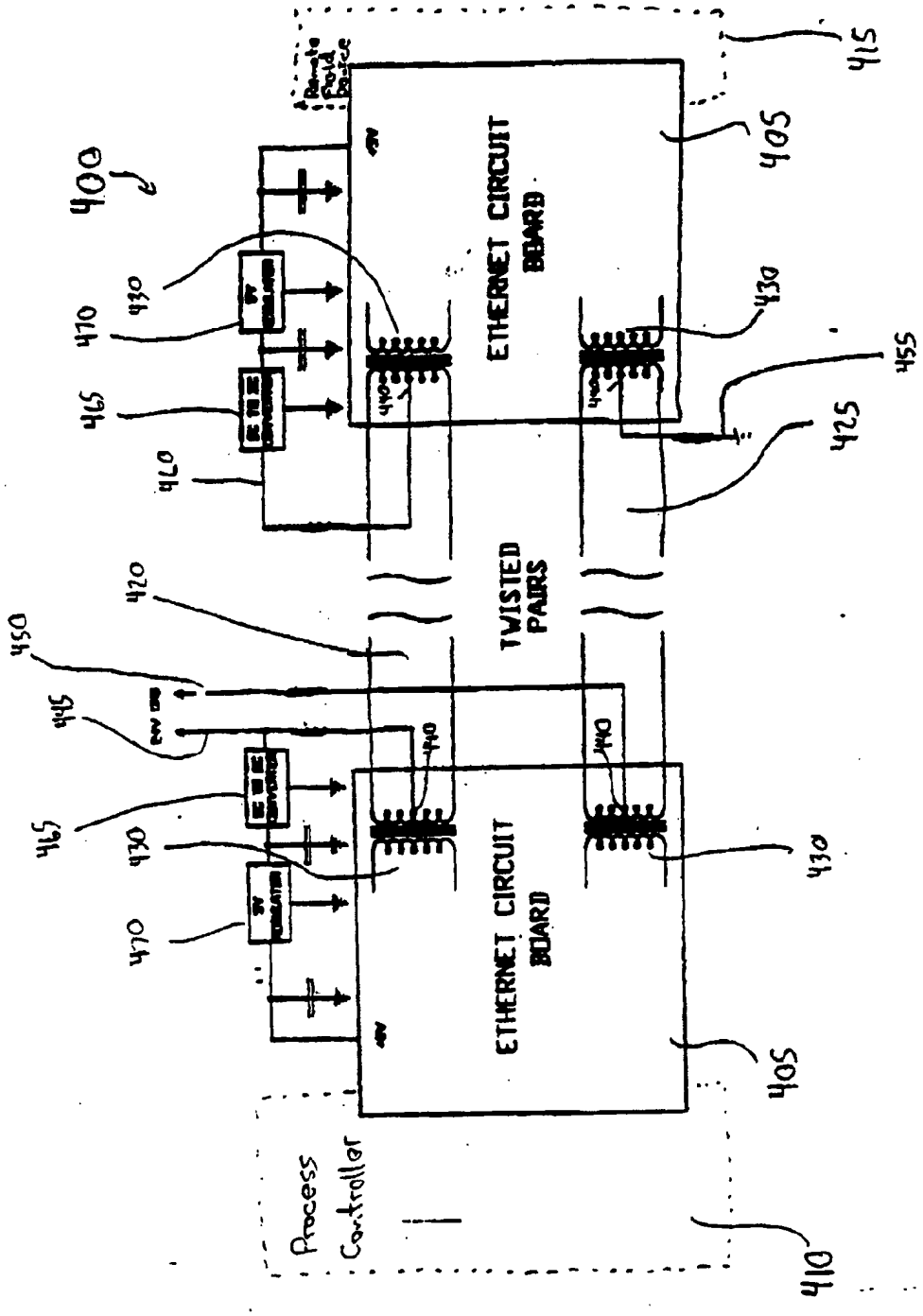


Fig. 4

Appendix

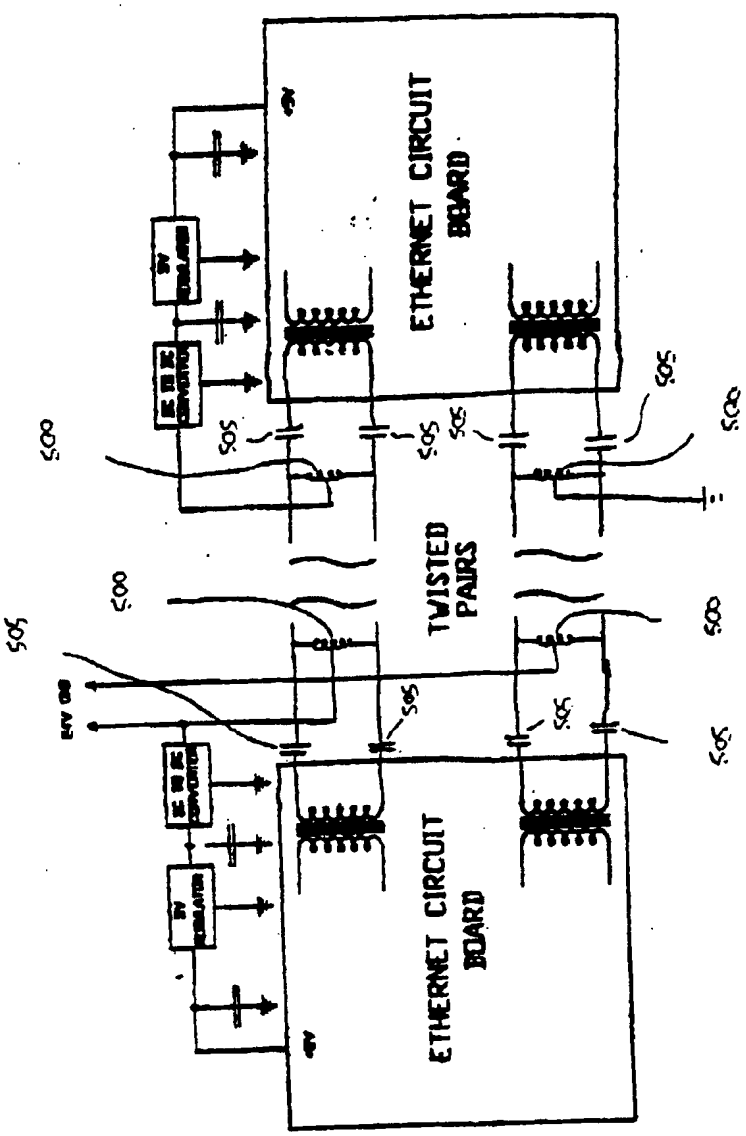


Fig. 5

Appendix

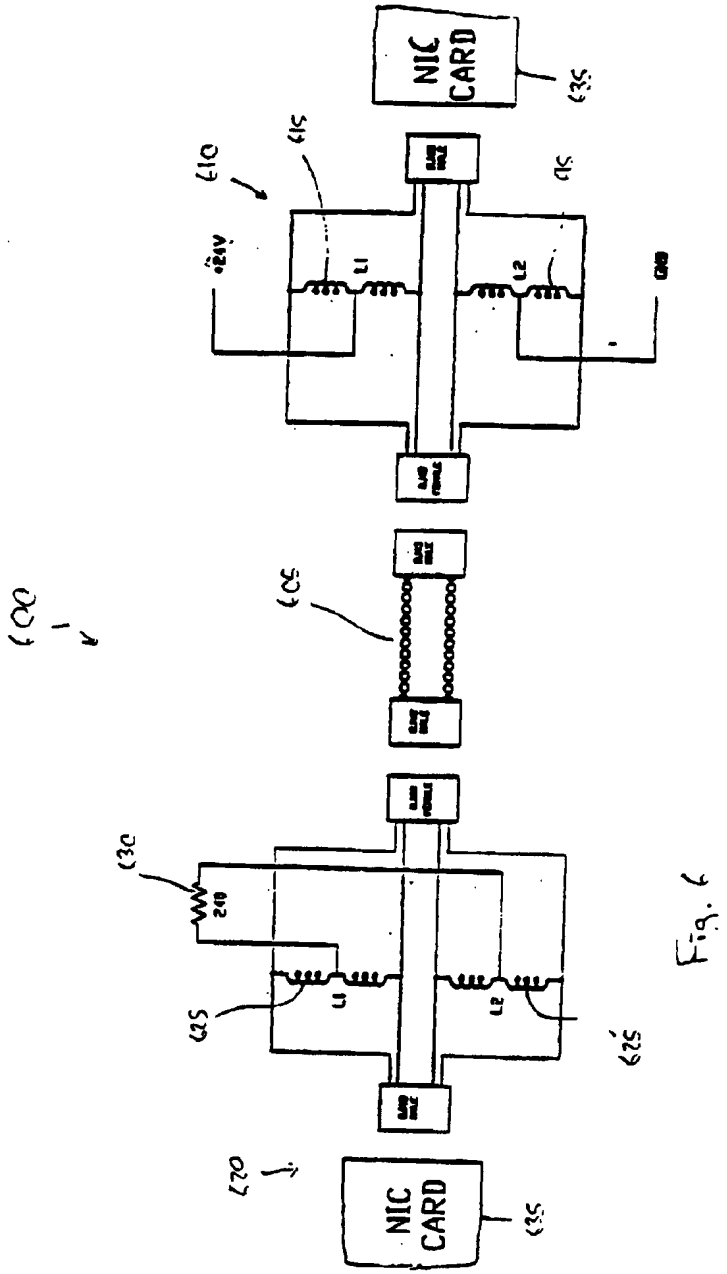


Fig. 6

Appendix

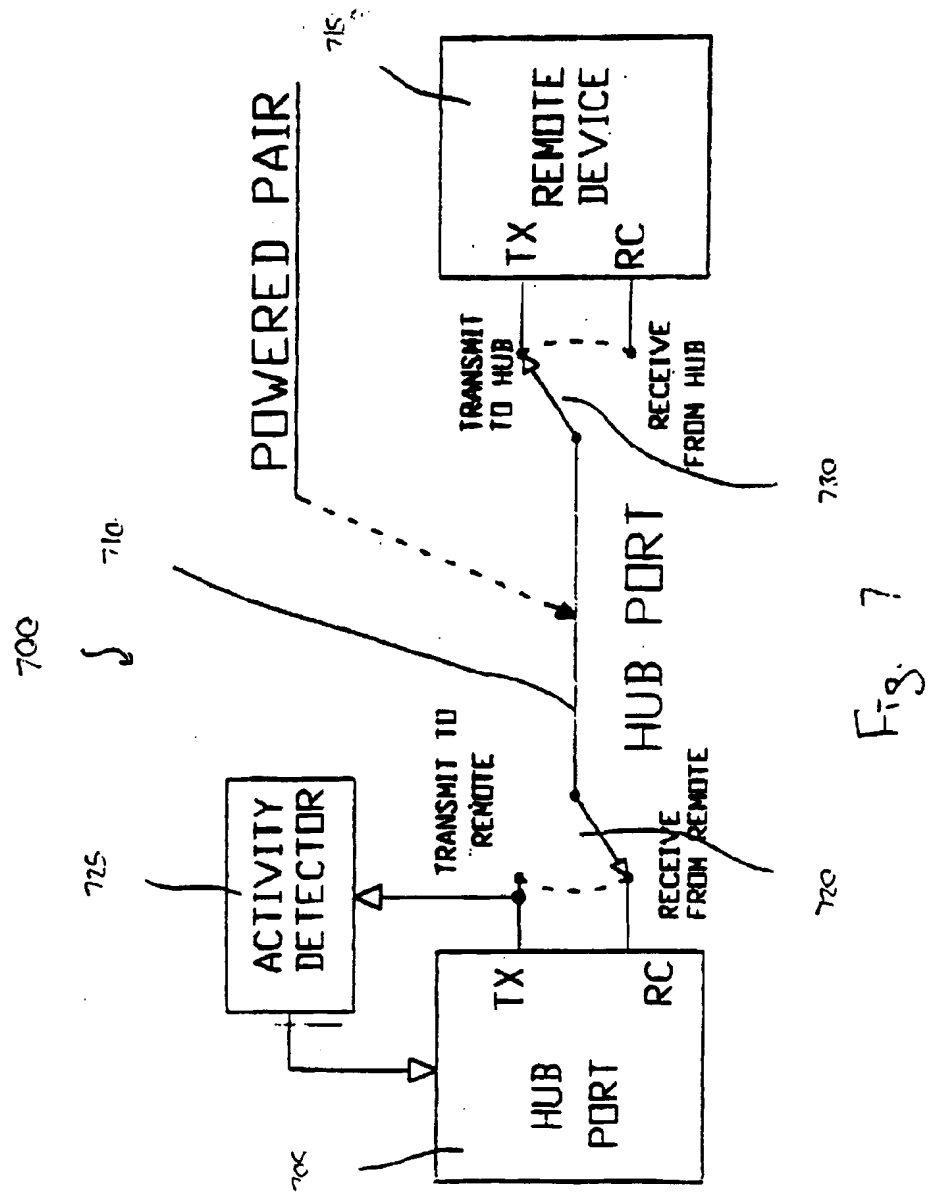


Fig. 7

Appendix

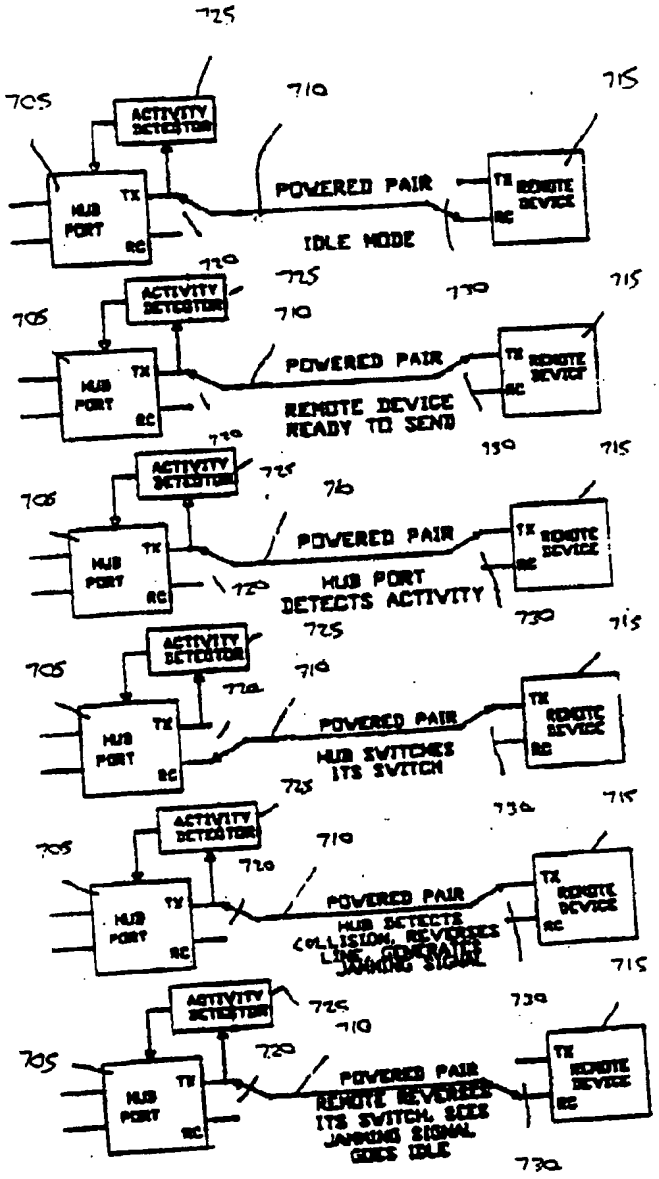


Fig. 8A

Fig. 8B

Fig. 8C

Fig. 8D

Fig. 8E

Fig. 8F

Appendix

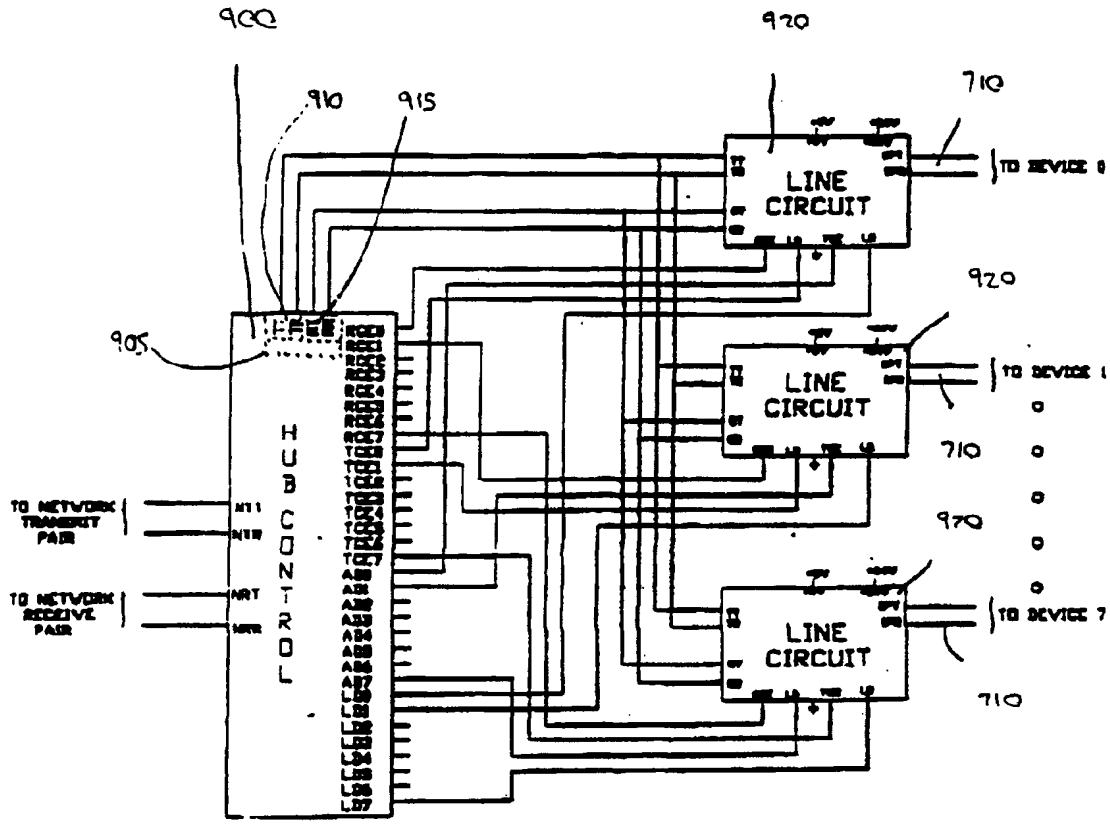


Fig. 9

Appendix

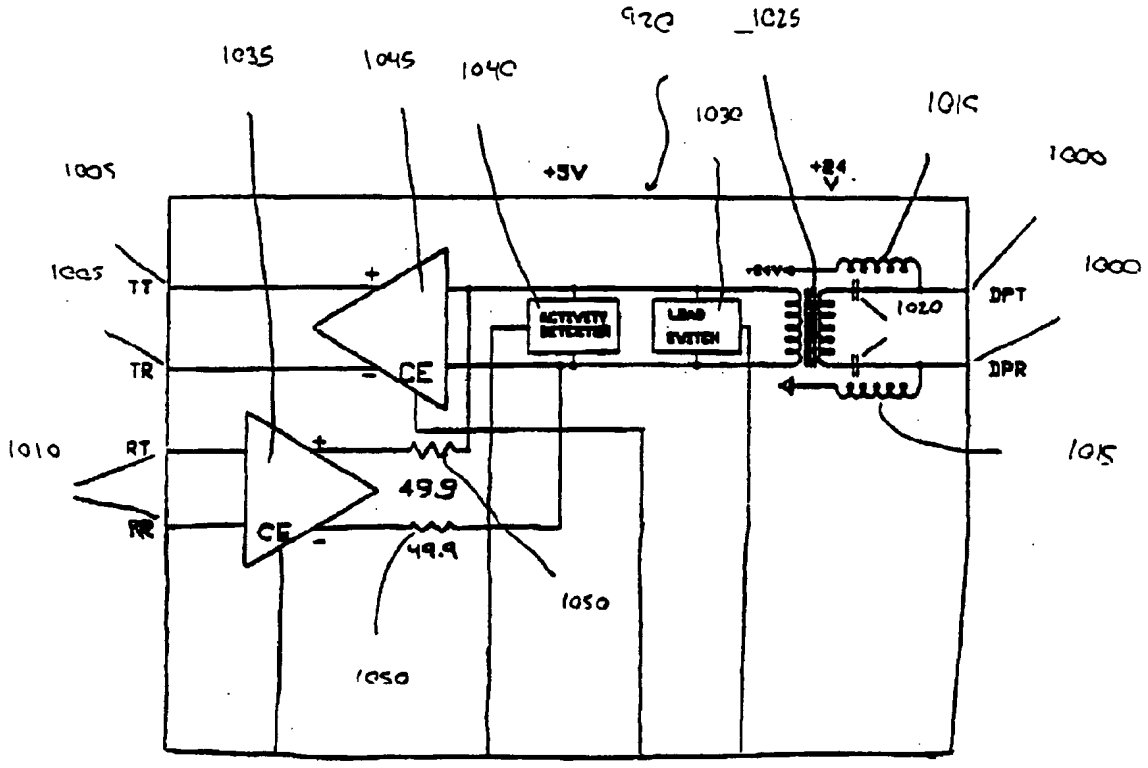


Fig. 10

Appendix

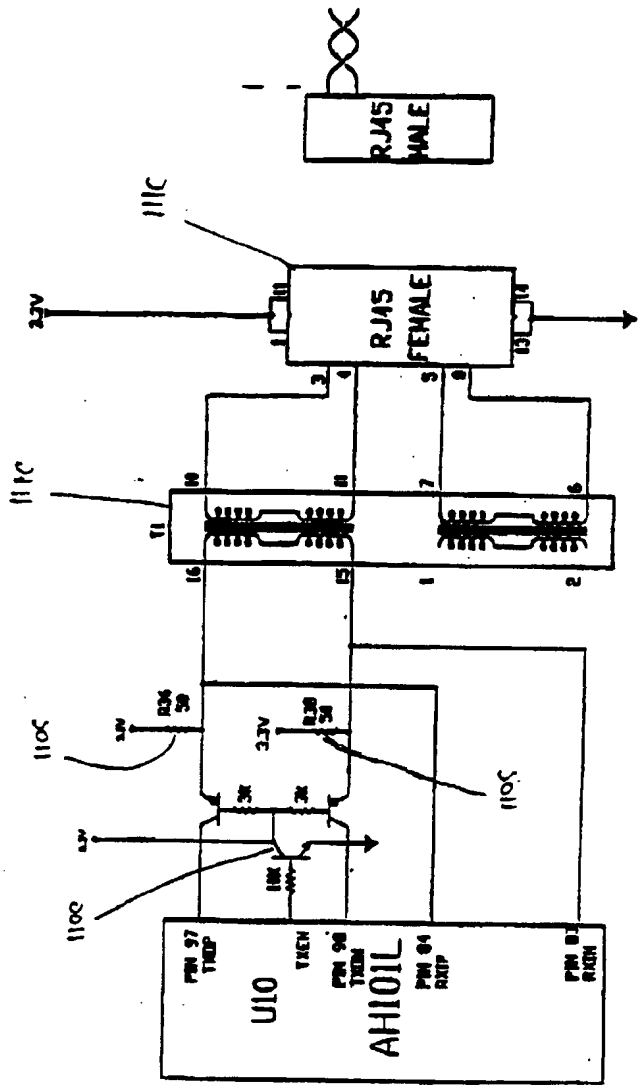


Fig. 11

Appendix

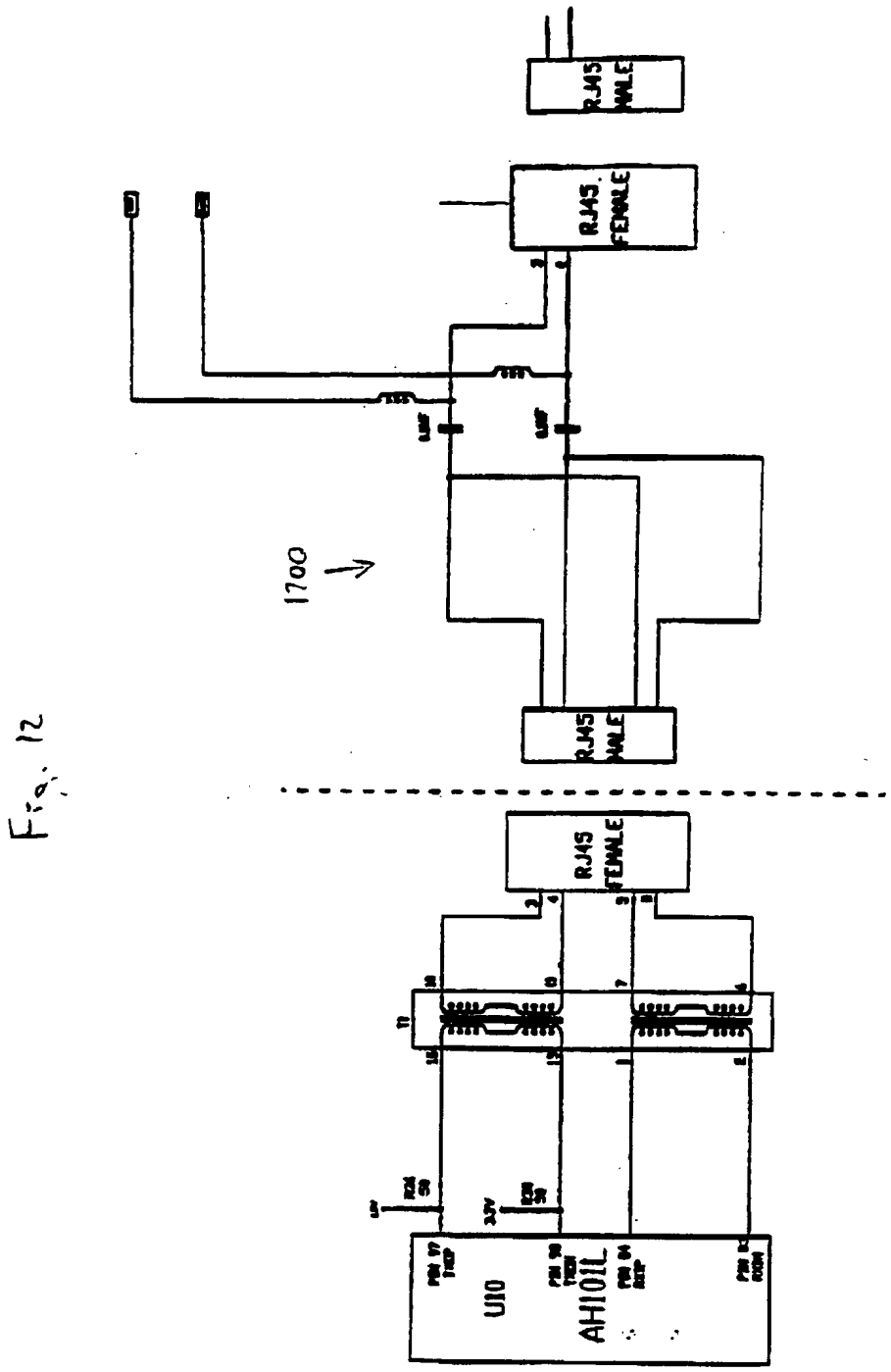


Fig. 12

Appendix

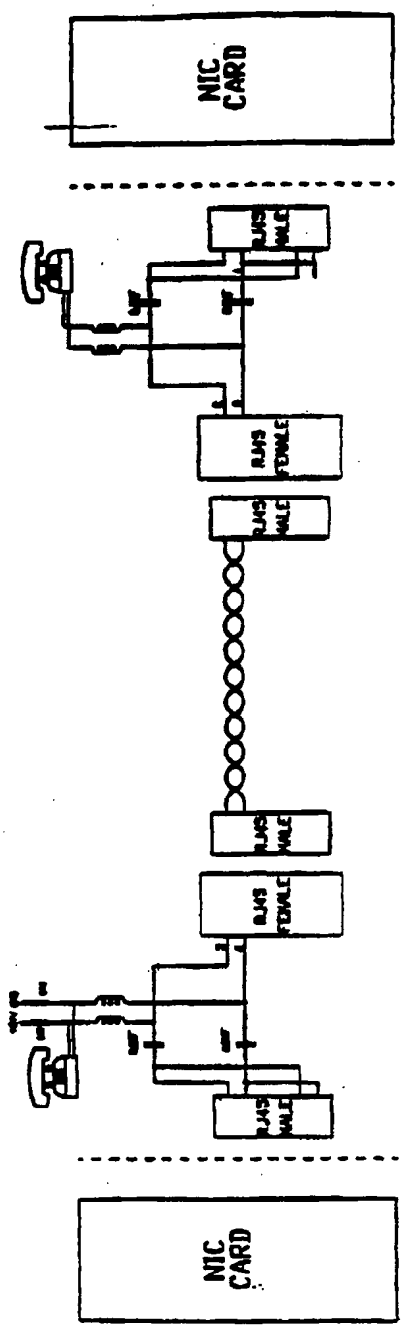
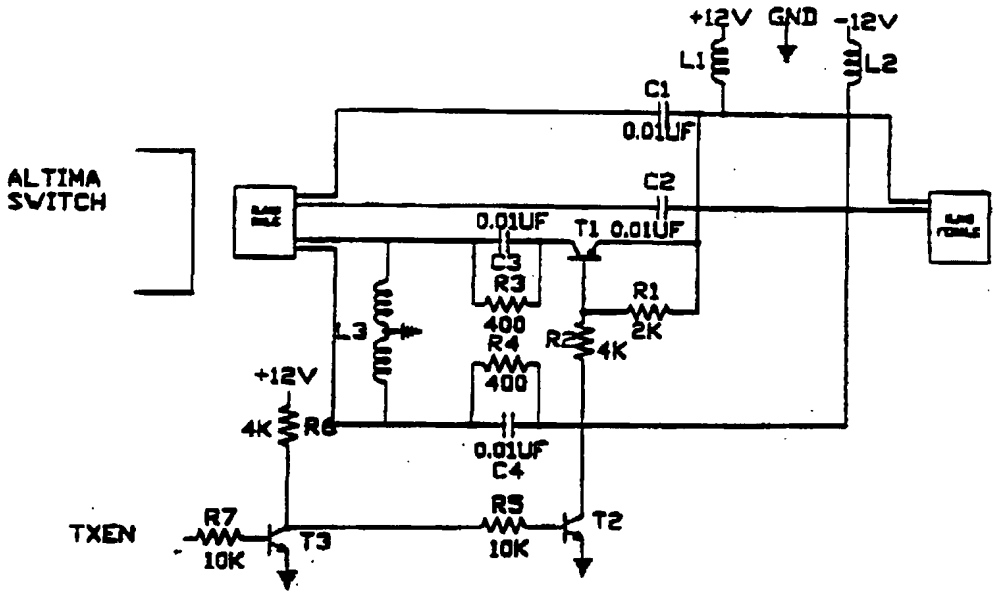


Fig. 17

Appendix

Fig. 14



Appendix

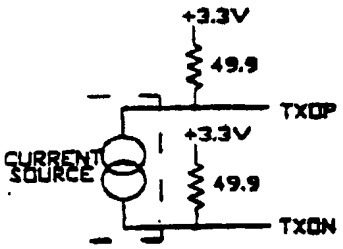


Fig. 15

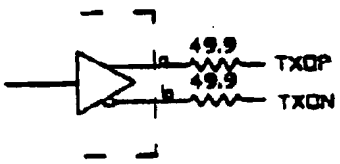


Fig. 16

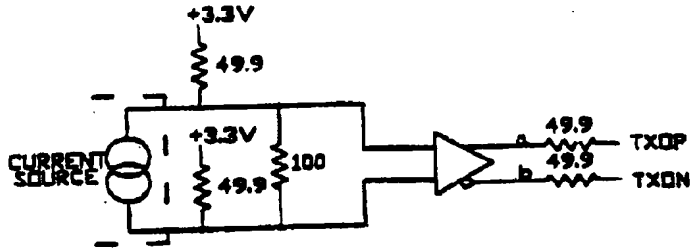


Fig. 17

Appendix

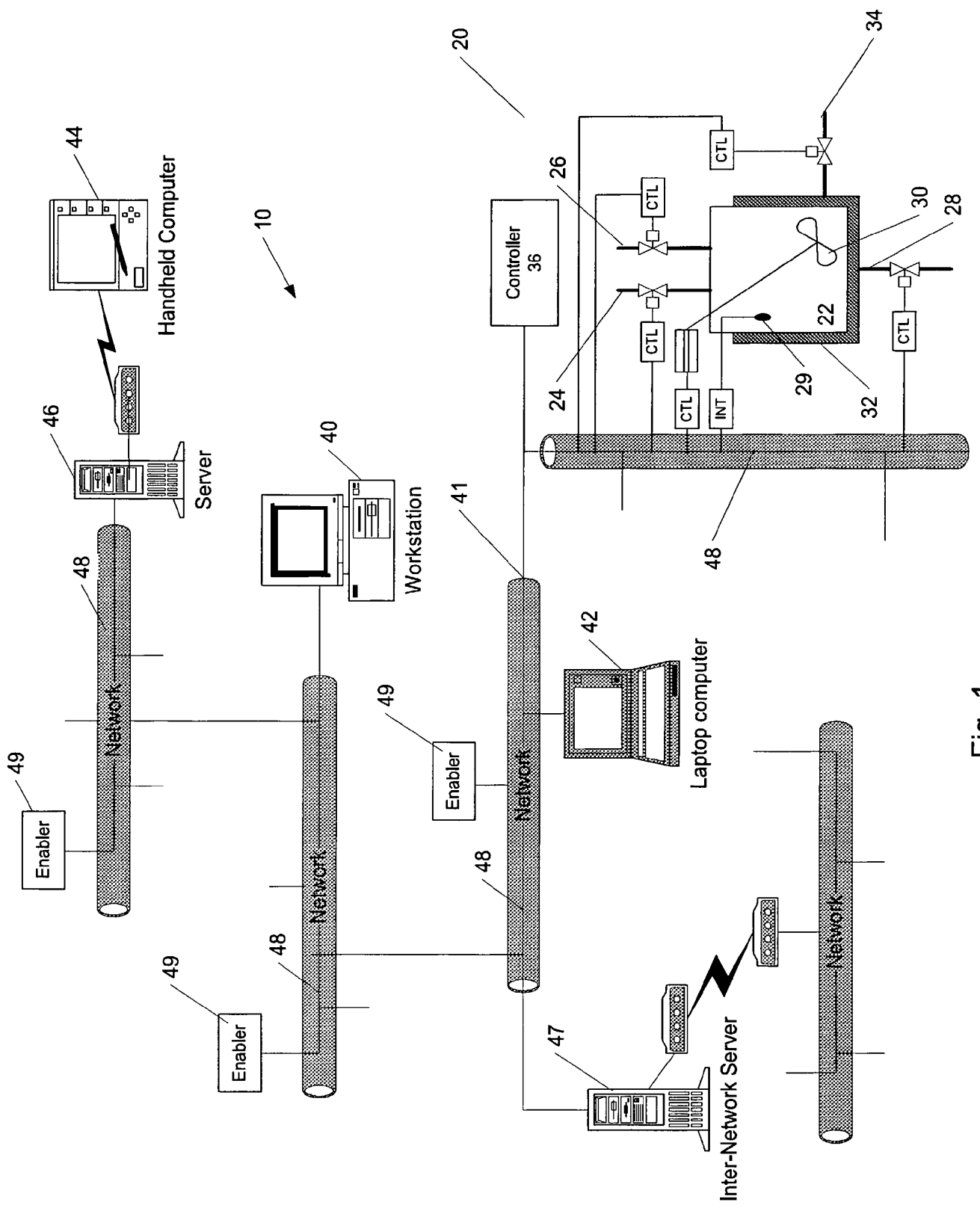


Fig. 1

Appendix

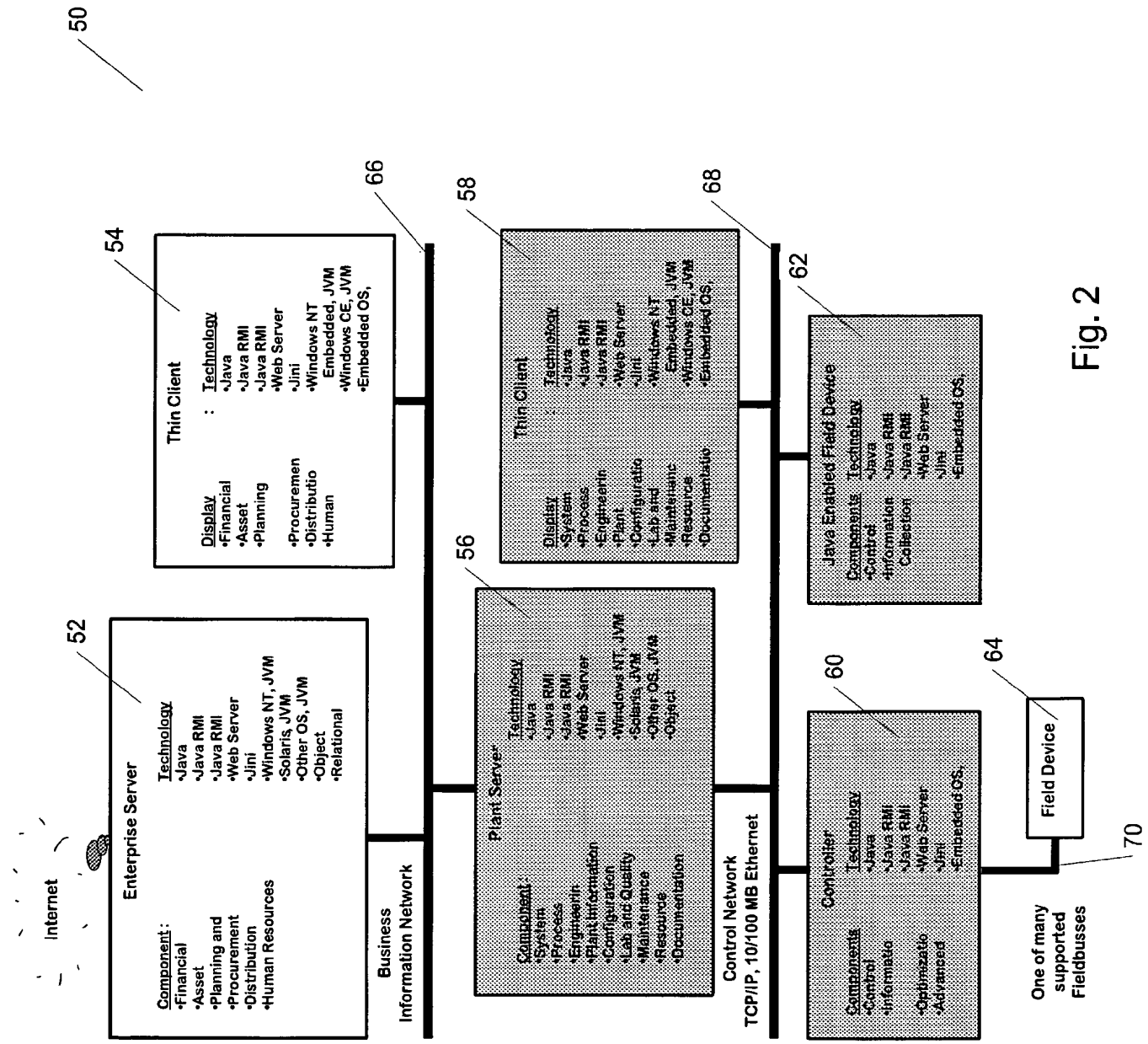


Fig. 2

Appendix

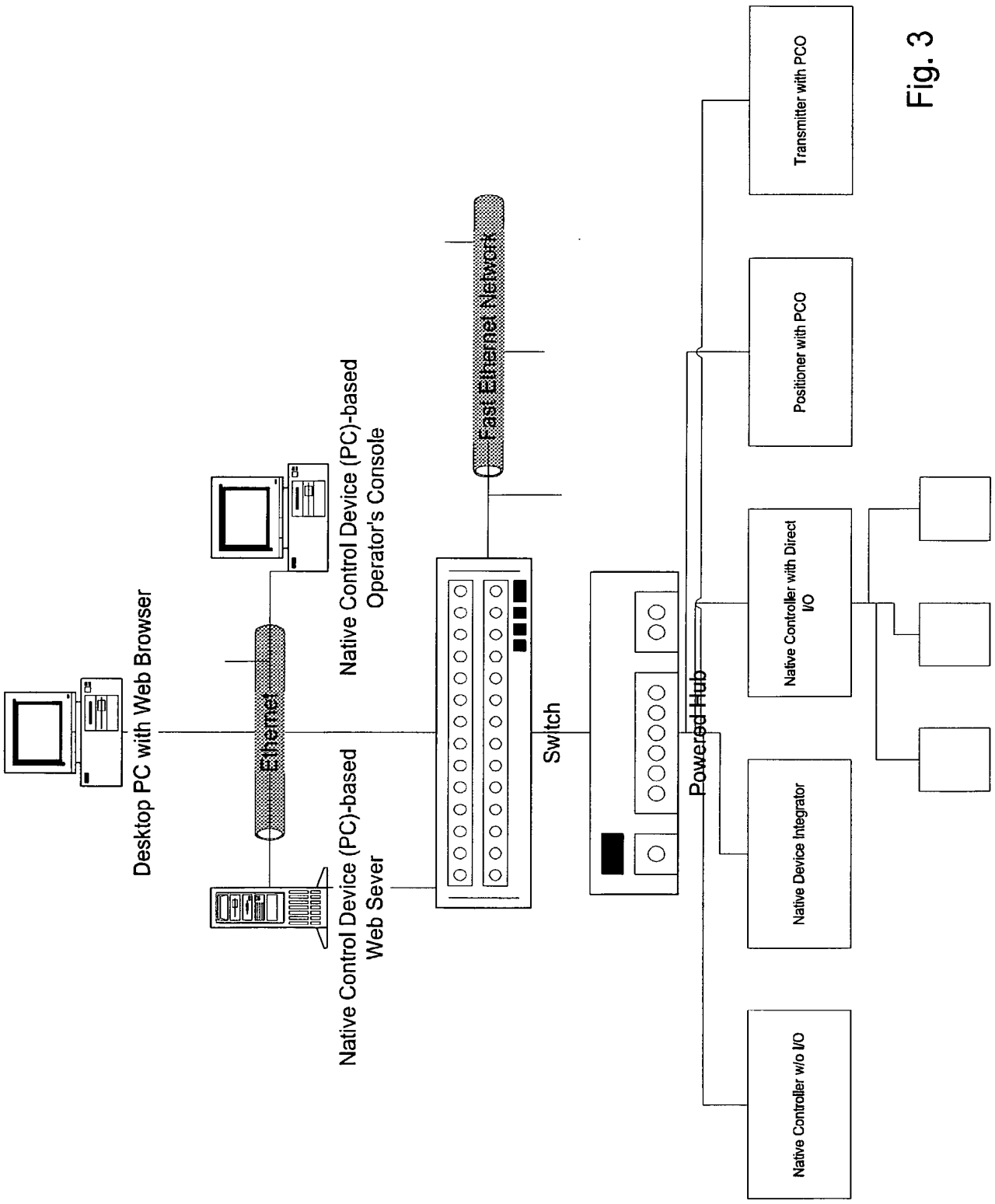
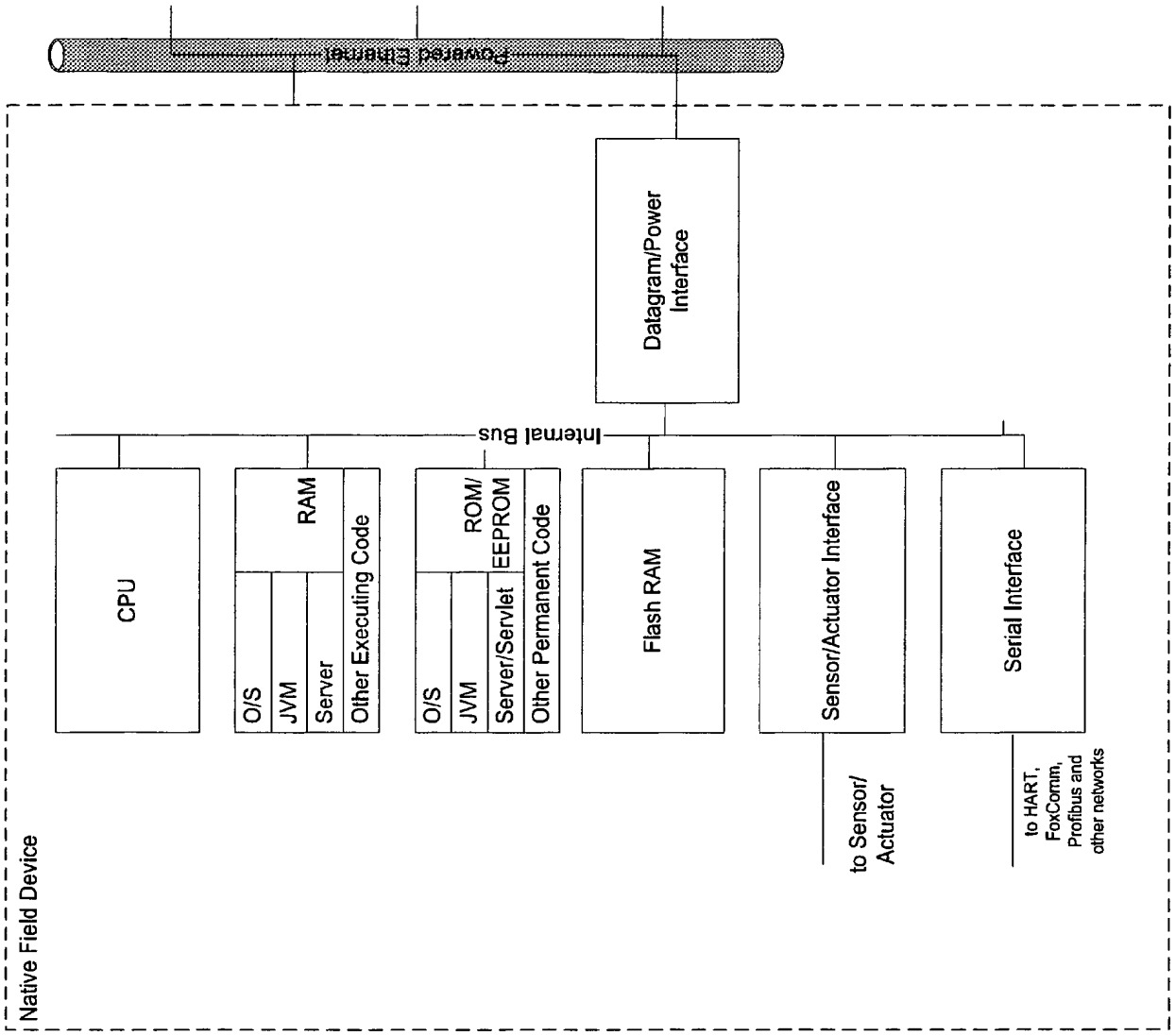


Fig. 3

Appendix

Fig. 4



Appendix

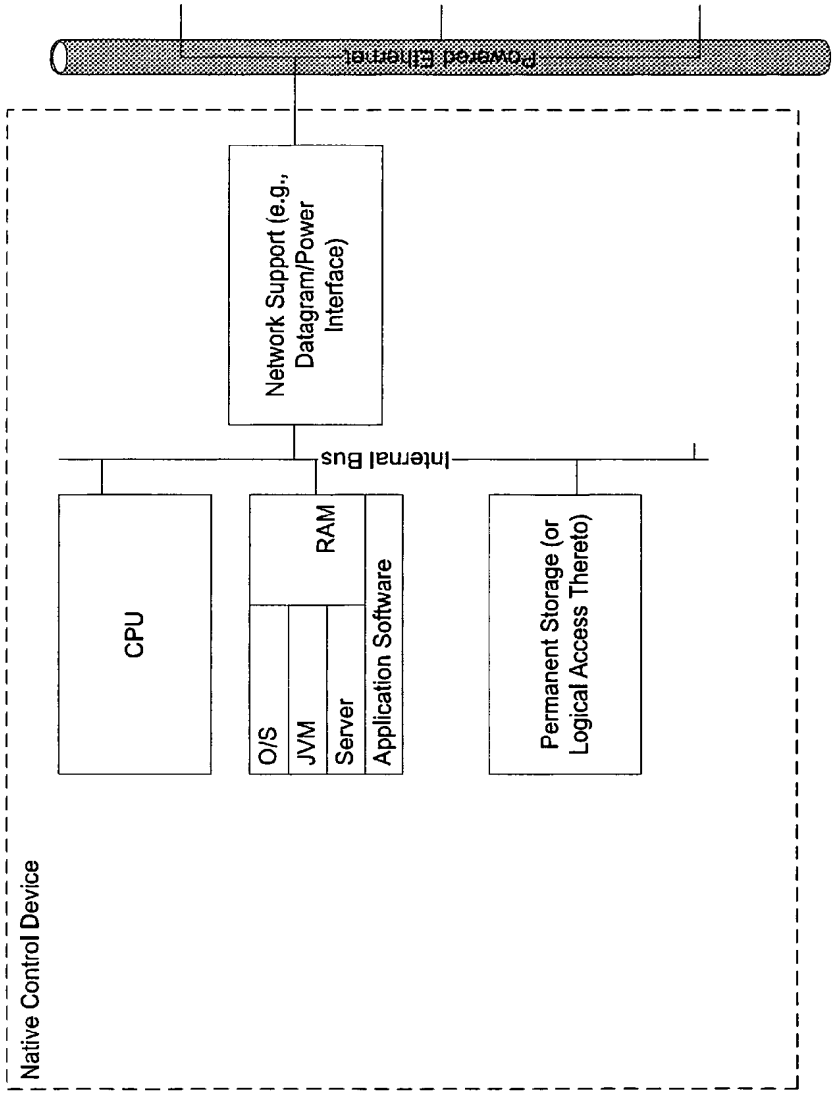


Fig. 5

Appendix

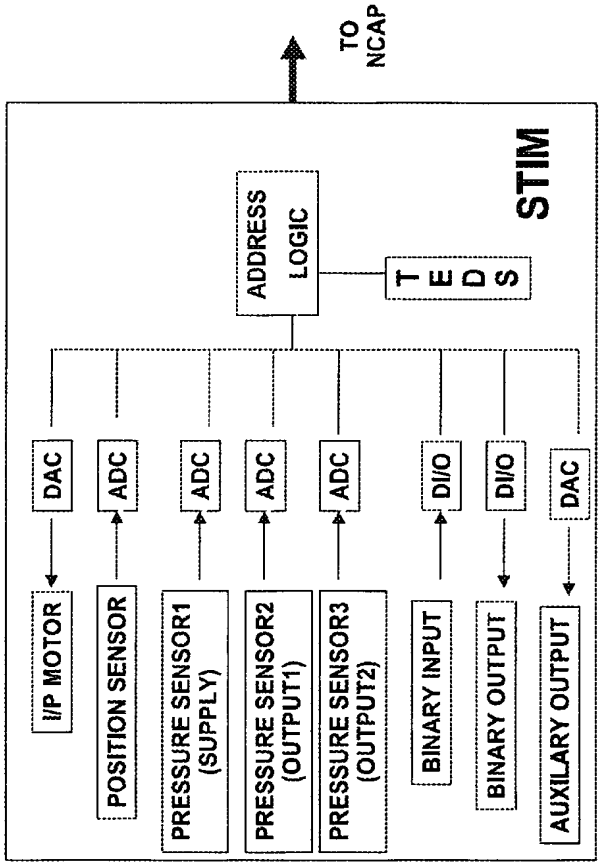


Fig. 6

Appendix

PROCESS CONTROL SYSTEM AND METHOD WITH IMPROVED DISTRIBUTION, INSTALLATION AND VALIDATION OF COMPONENTS

Background of the Invention

The invention pertains to control systems and, more particularly, to methods and apparatus for distributing, installing and/or validating components of such systems.

The terms "control" and "control systems" refer to the control of a device or system by monitoring one or more of its characteristics. This is used to insure that output, processing, quality and/or efficiency remain within desired parameters over the course of time. In many control systems, digital data processing or other automated apparatus monitor the device or system in question and automatically adjust its operational parameters. In other control systems, such apparatus monitor the device or system and display alarms or other indicia of its characteristics, leaving responsibility for adjustment to the operator.

Control is used in a number of fields. Process control, for example, is typically employed in the manufacturing sector for process, repetitive and discrete manufactures, though, it also has wide application in electric and other service industries. Environmental control finds application in residential, commercial, institutional and industrial settings, where temperature and other environmental factors must be properly maintained. Control is also used in articles of manufacture, from toasters to aircraft, to monitor and control device operation.

Digital data processing is firmly entrenched in the control systems. System designers increasingly rely on software to add functionality and flexibility to their systems. The installation and validation of process control system components, for example, has generally been quite simplistic. When a new hardware component is to be installed, an operator or test engineer must take the system offline, install the new component, bring the system back online, and monitor the operation of the component. If satisfied, the engineer makes the installation permanent.

Appendix

The I/A Series process control systems, manufactured by the assignee hereof, represent a major advance in this technology. They utilize a fault-tolerant architecture in which each control processor (CP), for example, has a redundant, shadow partner. Either of the partners can be replaced or updated while the system is still in operation. To this end, one of the CPs is placed in active mode, while its partner is being upgraded. The upgraded unit is then brought on-line, but only in shadow mode. Its operation is monitored by the engineer or operator, e.g., who compares its output with that of the active CP. If satisfied with the upgraded unit, the engineer can make it active, so that the remaining original CP can be upgraded. Redundant, fault-tolerant operation resumes once both CPs are similarly upgraded.

While the prior art techniques have proven effective to date, the ever increasing complexity of control systems render those techniques problematic. The physical replacement of hardware components, for example, can render maintenance unduly costly. This is exacerbated if the engineer must remain at a remote site until validation of the replacement is complete.

The replacement of software components is only somewhat less demanding. Apart from the aforementioned I/A Series systems, the prior art typically demands that an entire system be upgraded or, at least, taken off-line in order for an upgrade to be performed and tested. Moreover, the replacement of software components in prior art systems requires that the engineer be present at the replacement site, remaining there until the replacement is validated.

An object of this invention is to provide methods and apparatus that overcome these shortcomings. More particularly, an object is to provide improved methods and apparatus that facilitate the distribution, installation and validation of control systems and components.

A further object of the invention is to provide such methods and apparatus as facilitate the installation of components into active or on-line control systems.

A still further object of the invention is to provide such methods and apparatus as facilitate

Appendix

the distribution of control system components, e.g., from a manufacturer's site, and their installation, e.g., at a remote site.

Yet a still further object of the invention is to provide such methods and apparatus as can be readily implemented on existing digital data processing apparatus or special purpose control apparatus.

Still yet further related objects of the invention are to provide such methods and apparatus as can be applied in process control systems, environmental control systems, and the like.

Appendix

Summary of the Invention

The foregoing are among the objects attained by the invention, which provides in one aspect a control system with blocks or other components that facilitate validation of their own replacements. Further aspects of the invention provide control systems in which the components to be validated are downloaded and installed from a remote site, e.g., via e-commerce transaction.

Thus, in one aspect, the invention provides a control system that includes first and second control components, e.g., flow control objects for a process control system. The first component is coupled to a third control component, with which it transfers information, e.g., as part of an active or ongoing control process. The third component can be, for example, a temperature control object with which the first component (e.g., a flow control object) is in a cascaded arrangement. The second component can be, for example, an update or other potential replacement for the first component. Thus, for example, if the first component is a flow control object, the second component can be a similar control object with new or improved functionality.

The first and/or second components of a control system according to this aspect of the invention can effect substitution of the second component for the first. More particularly, they can effect coupling of the second component for information transfer with the third component and decoupling of the first component from such transfer with the third component. Preferably, such coupling and decoupling occur while the control system remains active.

According to a related aspect of the invention, prior to its substitution for the first component, the second component is coupled to receive information from the third component and/or any sources from which the first component receives information. The second component, however, is at least temporarily prevented from outputting information to any sinks to which the first component sends information. Instead, the output of the second component can be routed, along with that of the first component, to one or more comparators. These can reside elsewhere in the control system, e.g., in a supervisor object, or within the first and/or second components

Appendix

themselves. Substitution of the second component for the first rests on the success of comparison of their outputs and, in most applications, on confirmation by the operator or test engineer.

By way of example, a comparator can check the output of a newly installed flow control object against the output of an old object it is intended to replace. The comparator can notify the operator of the results of the comparison and, if the operator approves, the new object can be substituted for the old.

Further aspects of the invention provide a control system as described above in which one or more stores, e.g., pointers, symbols, variables, matrices, arrays, tables, records, databases, files, or other information stores, identify relationships between components and, more particularly, their respective sources and sinks. A list maintained in the first component, for example, can identify its various parameters that are sinks for the third component and, likewise, parameters in the third (or other components) for which it (the first component) is the source. Substitution of the second component for the first may be effected, for example, by replacing all connections to/from the first component with connections to/from the second component.

Still further aspects of the invention provide a control system as described above in which the first and second components are resident on a control processor, a "smart" field device, or on another digital data processor-based control device. The configured second component can be downloaded to that device, e.g., from a control system work station, while the control system is active and while the first component is operational, e.g., exchanging information with the third component. By way of further example, downloading from a manufacturer's site to the workstation can be effected as part of a contracted-for maintenance transaction, an upgrade transaction and/or as part of an e-commerce transaction, e.g., between the customer and the manufacturer. Such downloading, moreover, can be instigated by the customer or, automatically, by the first component or a software agent within the control system.

Yet still further aspects of the invention provide methods for control and methods for

Appendix

distributing and/or installing control components paralleling the operations described above.

Appendix

Brief Description of the Drawings

A more complete understanding of the invention may be attained by reference to the drawings, in which:

Figure 1 depicts a plurality of networked digital data processors for use in practicing the invention;

Figure 2 depicts objects for controlling a process in a system according to the invention; and

Figures 3A-3C depict the insertion and validation of a replacement object in the process control system of Figure 2.

Appendix

Detailed Description of the Illustrated Embodiment

Figure 1 depicts a digital data processing system of the type with which the invention may be practiced. The system includes a controller or other digital data processor 10 on which resides a process control system for monitoring or controlling a process 12. Though only one element is shown, those skilled in the art will appreciate that digital data processor 10 represents one or more workstations, controllers, microprocessors, embedded processors, "smart" field devices, or other digital data processing apparatus, utilized to control or monitor a process. Such digital data processing apparatus are of the types commercially available in the marketplace, operated in accord with the teachings herein to facilitate process control component installation and/or validation.

Medium 14 provides for transport, from site 16, of process control components -- and, more particularly, software aspects thereof -- that are to be installed and executed on digital data processor 10. Though illustrated to represent a LAN, WAN, or global network (Internet), those skilled in the art will appreciate that element 14 may represent any medium or mechanism through which software may be transported, electronically, physically or otherwise, from site 16 to digital data processor 10 or workstation 11.

Site 16 represents any source of software-based process control components or definitions thereof. This can include, for example, a retail store, warehouse or other distribution point of CDROMs, diskettes or other magnetic medium on which components or their definitions are stored. In a preferred embodiment, however, it represents a digital data processor that functions as a server, e.g., maintained by a manufacturer or other distributer, from which such components can be electronically transported to the digital data processor 10 or workstation 11. Without detracting from breadth of the teachings herein, site 16 is referred to hereinafter as a "site server."

Process 12 represents any industrial, manufacturing, service, environmental or other process amenable to monitoring or control (hereinafter, collectively, "control"). This is illustrated

Appendix

in greater detail in Figure 2, wherein a simple such process is shown as including valve 18 that governs the rate of fluid flow to aeration tank 20 which, in turn, transfers the liquid to storage tank 22. Sensors 24 and 26 monitor the state of process 12 and, thereby, facilitate its control by process control system 28 operating on the one or more digital data processors represented by element 10. Thus, sensor 24 is disposed in or adjacent to tank 20 for measuring the temperature of fluid therein, while sensor 26 measures the flow of fluid from aeration tank 20 to storage tank 22.

Figure 2 illustrates a sample process control system 28 in which the invention is employed. The system 28 includes three process control components 29, 30, 32 which, themselves, may include further components (not shown). Components 29, 30, 32 may comprise any combination of software and hardware features. In the illustrated embodiment only software features are shown -- here, as object-oriented programming (OOP) "objects." Other software constructs, by way of non-limiting example, DLL files, may be employed as well.

The workstation 11 or supervisor object 29 may initiate process control functions, including activation and execution of process control objects 30 and 32. The supervisor object 29 generates a temperature supervisory setpoint, e.g., based on operator input or a supervisory program. Object 30 serves as a temperature controller that utilizes a proportional-integral-derivative (PID) or other control algorithm to generate a flow remote setpoint based on the temperature setpoint from the supervisor object 29 and on temperature readings from sensor 24. Object 32 serves as a flow controller that, too, utilizes a PID or other control algorithm to generate a flow level based on the flow setpoint from object 30 and on flow readings from sensor 26. Objects 29, 30, 32 operate in the conventional manner known in the art, as modified in accord with the teachings herein to facilitate installation and/or validation of a further such component, e.g., replacement object 30a (Figure 3).

In process control terminology, supervisor 29 is referred to as a "source" for PID 30 and, more accurately, for the temperature setpoint parameter used by PID 30. Temperature sensor 24

Appendix

is also a source for PID 30. PID 32 is, conversely, referred to as a "sink" for PID controller 30 and, more accurately, for the flow setpoint parameter generated by it. Like terminology can be applied to the other elements and parameters that are sources (i.e., suppliers) or sinks (i.e., consumers) of information produced within the system 12.

The identities of the respective sources and sinks are maintained in centralized or distributed stores, e.g., pointers, symbols, variables, matrices, arrays, tables, records, databases, files, in the process control system 28. In one embodiment, for example, a centralized table (not shown), accessible by all of the components, stores those identities.

In a preferred embodiment, each element maintains information about its own sources and/or sinks. Thus, for example, in the illustrated embodiment, component 30 maintains pointers, addresses and/or identifiers of its various parameters that are sinks for information generated by the other components, e.g., elements 29 and 32. It also maintains pointers, addresses and/or identifiers of the components for which it (i.e., component 30) is a source. This facilitates reconnection of any components (e.g., 29, 32) that may be affected by replacement of component 30.

Figures 3A-3C depict a methodology for the insertion and validation of a replacement component in the process control system 28. Each drawing shows the system 28 in reduced format, using numeric labels to refer to the same-number blocks of Figure 2. The components of the system 28 may, as noted above, reside on one or more digital data processing apparatus 10.

In Figure 3A, a new or replacement component 30a is added to system 28, e.g., to the same digital data processor as resides component 30 which it (component 30a) will replace. In the illustrated embodiment, in which component 30a comprises an OOP object, an OOP class definition comprising an object template and methods is transmitted from the server 16 to a workstation 11, e.g., as part of a contracted-for maintenance transaction, upgrade transaction, or e-commerce transaction. There, an operator or engineer instantiates a replacement object 30a

Appendix

from the new class and configures the object for use in controlling process 12 via controller 10. Once configured, the object is downloaded to the controller, where it replaces a prior object 30 as described in further detail below.

Those skilled in the art will appreciate that the replacement object may be downloaded to digital data processor 10 via other mechanisms, as well. Thus, for example, the object definition or preconfigured object may be downloaded directly from server 16 to the digital data processor 10. Moreover, in embodiments that utilize non-OOP constructs, alternate data structures or code constructs (e.g., DDL files) may be downloaded to processor 10 directly or via workstation 11.

Referring back to Figure 1, depicted there are steps of an e-commerce transaction through which such a download can be effected. In step 1, an operator utilizes workstation 11 to send an inquiry to site server 16. This step, as well as the others required for installation and validation of the component 30a, preferably occur while the process control system 28 is online and operational, e.g., controlling process 12, and without substantive disruption or delay of any of the monitor and/or control functions performed by the replaced component, or of any components in communication therewith. As used herein, "substantive disruption or delay" refers to any disruption or delay having more than negligible impact on the aforesaid monitor and/or control functions.

In step 2, the site server 16 responds with information regarding possible upgrades. The server 16 can provide a complete listing of available upgrades or, alternatively, only those applicable to process control system 28.

In step 3, the operator selects a desired upgrade (e.g., the class for component 30a) and provides requisite purchase account information, e.g., credit card, PO number, etc. The operator also supplies whatever additional information is required or desirable in order to effect the download from the server 16.

Appendix

In step 4, the site server 16 downloads the replacement software to workstation 11, e.g., in the form of a .java file, a .class file, a DLL file (e.g., for non-OOP implementations), or in any other format suitable for adding software aspects of component 30a to workstation 11.

In step 5, workstation 11 instantiates and configures component 30a, based on the new class, to be similar to component 30 and downloads it for testing in the control processor 10.

Those skilled in the art will, of course, appreciate that numerous other alternatives may be employed to add the class for component 30a (or, e.g., in non-object oriented systems, to add the component 30a, itself) to digital data processor 10. These include, for example, installing the component from a CDROM, diskette, or other medium. By way of further example, electronic downloading of the component 30a can be "requested" by the system 28, itself. For example, the component 30 being replaced can itself query the site server 16 for upgrades, e.g., periodically, upon expiration of an obsolescence timer, in response to messaging from site server 12, or otherwise.

Turning to Figure 3B, the downloaded component 30a is coupled to the sources of the block 30a that it is intended to replace, i.e., block 30. In a preferred embodiment, the component is downloaded in the form of a JAVA ".class" file and, hence, it is immediately operational for purposes for such coupling. In other embodiments, additional steps (such as compilation, linking/loading, etc.) required to bring component 30a into existence on digital data processor 10 and/or to make it available for coupling into control system 28 can be effected at this time.

The manner in which component 30a is coupled to the sources of component 30 varies in accord with the manner in which source information is stored in process control system 28. For example, if pointers to sources for component 30 are maintained in its own stores, component 30a can copy that information. Alternatively, if source information is coded into component 30 via a configurator (not shown), such a configurator may be employed to imbue component 30a with the same information.

Appendix

In addition to such "source coupling," the outputs of blocks 30 and 30a are routed to a comparator (labeled "=") so that they can be compared. This can reside within supervisor object 29, within components 30, 30a themselves, within workstation 11, or elsewhere within the system 28. Routing can be effected by adding additional sinks to blocks 30, 30a, i.e., by defining them as sources for the comparator, or otherwise. Apart from routing its outputs to the comparator, replacement object 30a is temporarily prevented from applying those outputs to other components, e.g., 29, 32, in the system.

Once coupled as described above, the potential replacement block 30a is tested to determine whether its output is comparable with that of the block 30. To this end, block 30a processes inputs received identically with block 30 and generates output comparable with that of block 30. Comparison can be performed in any manner known in the art, preferably, using comparison methodologies defined in the newly instantiated object 30a, in the original block 30, or elsewhere in the system. In the illustrated embodiment, graphical or other output indicative of the comparison is generated by the comparator for routing to a log and/or to the operator workstation 11.

If the results of the comparison are acceptable, and if the operator signals his or her confirmation, installation of the replacement module is completed as shown in Figure 3C. This is effected by further modification of the centralized or distributed source/sink stores so that component 30a is identified as the source or sink of any component 29, 32 for which component 30 was previously so identified. Information regarding component 30 can then be removed from those stores. As with the preceding steps, this too preferably occurs while the process control system 28 is operational so as "not to miss a beat."

In one embodiment, the foregoing operations are effected by executing sequences of the type that follow on the workstation 11 and control processor 10:

Workstation

Appendix

1. Obtain new composite, block, or part class from server 16.
2. Instantiate new object 30a from class.
3. Instantiate all external bi-directional (cascade) input variables.
4. Configure new object 30a to handle tasks of object it is to replace:
    - Connect forward parameter of all cascade inputs and connect variable of all uni-directional inputs to source of inputs of old object.
    - Connect back parameter of all cascade outputs to back parameter of existing outputs.
    - Create a sinkList object for the list of output sinks of object 30 being replaced.
    - Create test runList object.
    - Create final runList object.
5. Serialize new object 30a, sinkList object, test runList object, and final runList object to an object file for each.

Control Processor

1. Download new class files.
2. Download new object 30a and new test runList object file.
3. Send message with object names.
4. Set flag in application to instantiate new object 30a from the new class.
5. Read object file to customize object.
6. Replace existing runList with new runList from runList object file.
7. Continue executing runList.
8. Bring output of old and new objects to display with trends.
9. Let operator experiment by creating transients.
10. If new object performs satisfactorily, the operator pushes the "accept" button.
11. Download sinkList object for the list of output sinks of original object 30 and make new output connections in sink objects.
12. Disconnect (null) inputs in old object 30.
13. Download final runList and continue execution.
14. Change cascade input from forward parameter to variable.
15. Delete (null) old object 30 and object files.

Described above are methods and apparatus achieving the desired objects. Those skilled in the art will appreciate that the embodiments described herein and shown in the drawings are examples of the invention and that other embodiments incorporating one or more of the mechanisms and techniques herein, or equivalents thereof, fall within the scope of the invention.

Appendix

Thus, for example, further embodiments of the invention provide environmental control systems utilizing apparatus and methods like those herein to monitor and/or control heating, ventilation, cooling, and other environmental factors. Yet still further embodiments of the invention provide industrial control systems, manufacturing control systems, or the like, that also utilize apparatus and methods like those herein to monitor and/or control respective industrial, manufacturing or other processes.

By way of further non-limiting example, it will be appreciated that the comparison phase can involve matching the output of potential replacement object 30a with a standard other than the output of object 30.

By way of still further non-limiting example, it will be appreciated that replacement components can be downloaded directly to a controller or other digital data processing apparatus in which they are to be executed.

In view of the foregoing, what is claimed is:

Appendix

1. A control system, including

a first component and a second component, the first component being coupled to and transferring information with a third component, and

at least a selected one of the first and second components selectively decoupling the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

2. A control system according to claim 1, wherein the control system remains operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

3. A control system according to claim 1, wherein the second component generates information for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

4. A control system according to claim 3, comprising a comparator that is coupled with the first and second components and that compares the information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

5. A control system according to claim 4, wherein the comparator is coupled to an operator console and wherein the comparator applies thereto an output indicative of the results of the comparison.

6. A control system according to claim 4, wherein the comparator is resident in any of the first component, second component, third component, an operator workstation, elsewhere in the control system.

Appendix

7. A control system according to claim 4, wherein the selected component selectively decouples the first and third components from information transfer and couples the second and third components for information transfer in response to any of an operator command and a result of the comparison.

8. A control system according to claim 1, comprising one or more stores identifying any of sources and sinks of information transferred between the components.

9. A control system according to claim 8, wherein the one or more stores are any of centralized and distributed within the control system.

10. A control system according to claim 9, wherein the one or more stores comprise any of pointers, symbols, variables, matrices, arrays, tables, records, databases, and files.

11. A control system according to claim 8, wherein at least one of the first and second components updates one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

12. A control system, including

a first component coupled to and transferring information with a third component, at least the first component being resident on a first digital data processor,

a second digital data processor coupled with the first digital data processor,

the second digital data processor transferring any of a second component and a definition thereof to the first digital data processor,

Appendix

at least a selected one of the first and second components selectively decoupling the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

13. A control system according to claim 12, wherein the control system remains operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

14. A control system according to claim 12, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor in response to a request from an operator.

15. A control system according to claim 12, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor as part of any of a contracted-for maintenance transaction, an upgrade transaction and an e-commerce transaction.

16. A control system according to claim 12, wherein the second component generates information for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

17. A control system according to claim 16, comprising a comparator that is coupled with the first and second components and that compares the information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

18. A control system according to claim 17, wherein the selected component selectively decouples the first and third components from information transfer and couples the second and third components for information transfer in response to any of an operator command

Appendix

and a result of the comparison.

19. A control system according to claim 12, comprising one or more stores identifying any of sources and sinks of information transferred between the components.

20. A control system according to claim 19, wherein at least one of the first and second components updates one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

21. A control system that controls any of a manufacturing, industrial, and environmental control process, the control system including

a first component and a second component, the first component being coupled to and transferring information with a third component,

at least one of the first and third components being arranged for at least one of monitoring and controlling said manufacturing, industrial, and environmental control process,

at least a selected one of the first and second components selectively decoupling the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

22. A control system according to claim 21, wherein the control system remains operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

23. A control system according to claim 21, wherein at least one of the first and third components is directly or indirectly coupled to a device that at least one of monitors and

Appendix

controls said manufacturing, industrial, and environmental control process.

24. A control system according to claim 21, wherein the second component generates information for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

25. A control system according to claim 24, comprising a comparator that is coupled with the first and second components and that compares the information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

26. A control system according to claim 25, wherein the comparator is coupled to an operator console and wherein the comparator applies thereto an output indicative of the results of the comparison.

27. A control system according to claim 25, wherein the comparator is resident in any of first component, second component, third component, an operator workstation, elsewhere in the control system.

28. A control system according to claim 25, wherein the selected component selectively decouples the first and third components from information transfer and couples the second and third components for information transfer in response to any of an operator command and a result of the comparison.

29. A control system according to claim 21, comprising one or more stores identifying any of sources and sinks of information transferred between the components.

30. A control system according to claim 29, wherein the one or more stores are any of centralized and distributed within the control system.

Appendix

31. A control system according to claim 30, wherein the one or more stores comprise any of pointers, symbols, variables, matrices, arrays, tables, records, databases, and files.

32. A control system according to claim 29, wherein at least one of the first and second components updates one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

33. A control system that controls any of a manufacturing, industrial, and environmental control process, the control system including,

a first component coupled to and transferring information with a third component, at least the first component being resident on a first digital data processor,

at least one of the first and third components being arranged for at least one of monitoring and controlling said manufacturing, industrial, and environmental control process,

a second digital data processor coupled with the first digital data processor,

the second digital data processor transferring any of a second component and a definition thereof to the first digital data processor,

at least a selected one of the first and second components selectively decoupling the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

34. A control system according to claim 33, wherein the control system remains operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

Appendix

35. A control system according to claim 33, wherein at least one of the first and third components is directly or indirectly coupled to a device that at least one of monitors and controls said manufacturing, industrial, and environmental control process.

36. A control system according to claim 33, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor in response to a request from an operator.

37. A control system according to claim 33, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor as part of any of a contracted-for maintenance transaction, an upgrade transaction and an e-commerce transaction.

38. A control system according to claim 33, wherein the second component generates information for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

39. A control system according to claim 38, comprising a comparator that is coupled with the first and second components and that compares the information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

40. A control system according to claim 39, wherein the selected component selectively decouples the first and third components from information transfer and couples the second and third components for information transfer in response to any of an operator command and a result of the comparison.

41. A control system according to claim 33, comprising one or more stores identifying any of sources and sinks of information transferred between the components.

Appendix

42. A control system according to claim 41, wherein at least one of the first and second components updates one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

43. A method of operating a control system of the type having a first component that is coupled to and that transfers information with a third component, the method comprising

providing a second component,

selectively operating any of the first and second components to decouple the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

44. A method according to claim 43, including the step of keeping the control system operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

45. A method of operating a control system according to claim 43, comprising generating information with the second component for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

46. A method of operating a control system according to claim 45, comprising comparing information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

47. A method of operating a control system according to claim 46, comprising applying to an

Appendix

operator console an output indicative of the results of the comparison.

48. A method of operating a control system according to claim 46, comprising responding to any of an operator command and a result of the comparing step by selectively decoupling the first and third components from information transfer and, in lieu thereof, coupling the second and third components for information transfer.

49. A method of operating a control system according to claim 43, comprising providing one or more stores identifying any of sources and sinks of information transferred between the components.

50. A method of operating a control system according to claim 49, comprising updating one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

51. A method of operating a control system of the type having

a first component coupled to and transferring information with a third component, at least the first component being resident on a first digital data processor,.

the method comprising

transferring, from a second digital data processor to the first digital data processor, any of a second component and a definition thereof,

operating any of the first and second components to selectively decoupling the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

Appendix

52. A method according to claim 51, including the step of keeping the control system operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

53. A method of operating a control system according to claim 51, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor in response to a request from an operator.

54. A method of operating a control system according to claim 51, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor as part of any of a contracted-for maintenance transaction, an upgrade transaction and an e-commerce transaction.

55. A method of operating a control system according to claim 51, generating information with the second component for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

56. A method of operating a control system according to claim 55, comprising comparing information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

57. A method of operating a control system according to claim 56, comprising responding to any of an operator command and a result of the comparing step by selectively decoupling the first and third components from information transfer and coupling the second and third components for information transfer.

58. A method of operating a control system according to claim 57, comprising providing one

Appendix

or more stores identifying any of sources and sinks of information transferred between the components.

59. A method of operating a control system according to claim 58, comprising updating one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

60. A method of operating a control system that controls any of a manufacturing, industrial, and environmental control process, the control system including

a first component that is coupled to and that transfers information with a third component,

at least one of the first and third components being arranged for at least one of monitoring and controlling said manufacturing, industrial, and environmental control process,

the method comprising

providing a second component,

operating of any of the first and second components to selectively decouple the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

61. A method according to claim 60, including the step of keeping the control system operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

62. A method of operating a control system according to claim 60, wherein at least one of the first and third components is directly or indirectly coupled to a device that at least one of

Appendix

monitors and controls said manufacturing, industrial, and environmental control process.

63. A method of operating a control system according to claim 60, generating information with the second component for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

64. A method of operating a control system according to claim 63, comprising comparing information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

65. A method of operating a control system according to claim 64, comprising applying to an operator console an output indicative of the results of the comparison.

66. A method of operating a control system according to claim 64, comprising responding to any of an operator command and a result of the comparing step by selectively decoupling the first and third components from information transfer and coupling the second and third components for information transfer.

67. A method of operating a control system according to claim 60, comprising providing one or more stores identifying any of sources and sinks of information transferred between the components.

68. A method of operating a control system according to claim 60, comprising updating one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

69. A method of operating a control system that controls any of a manufacturing, industrial,

Appendix

and environmental control process, the control system including

a first component coupled to and transferring information with a third component, at least the first component being resident on a first digital data processor,

at least one of the first and third components being arranged for at least one of monitoring and controlling said manufacturing, industrial, and environmental control process,

the method including

transferring, from a second digital data processor to the first digital data processor, any of a second component and a definition thereof,

operating any of the first and second components to selectively decouple the first and third components from information transfer and, in lieu thereof, coupling the second component to the third component for transfer of information therewith.

70. A method according to claim 69, including the step of keeping the control system operational while the first and third components are decoupled from information transfer and wherein the second and third components are coupled for information transfer.

71. A method of operating a control system according to claim 69, wherein at least one of the first and third components is directly or indirectly coupled to a device that at least one of monitors and controls said manufacturing, industrial, and environmental control process.

72. A method of operating a control system according to claim 69, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor in response to a request from an operator.

Appendix

73. A method of operating a control system according to claim 69, wherein the second digital data processor transfers any of the second component and the definition thereof to the first digital data processor as part of any of a contracted-for maintenance transaction, an upgrade transaction and an e-commerce transaction.

74. A method of operating a control system according to claim 69, generating information with the second component for possible transfer to the third component, which information is comparable with information generated by the first component for transfer to such third component.

75. A method of operating a control system according to claim 74, comprising comparing information generated by the first component for transfer to such third component with information generated by the second component for possible transfer to the third component.

76. A method of operating a control system according to claim 75, comprising responding to any of an operator command and a result of the comparing step by selectively decoupling the first and third components from information transfer and coupling the second and third components for information transfer.

77. A method of operating a control system according to claim 69, comprising providing one or more stores identifying any of sources and sinks of information transferred between the components.

78. A method of operating a control system according to claim 69, comprising updating one or more of the stores in order to decouple the first and third components from information transfer and in order to couple the second and third components for information transfer.

79. A method of electronic commerce comprising

Appendix

transferring, from a first digital data processor to a second digital data processor, any of a first control system component and a definition thereof,

the first control system component being adapted for use in a control system of the type having a second control system component coupled to and transferring information with a third component,

any of the first and second control system components being further adapted for selectively decoupling the second and third components from information transfer and, in lieu thereof, coupling the first control component to the third component for transfer of information therewith

80. A method according to claim 79, comprising transferring the first control system component from the first digital data processor to the second digital data processor as part of any of a contracted-for maintenance transaction, an upgrade transaction and an e-commerce transaction.

81. A method according to claim 79, comprising transferring the first control system component from the first digital data processor to the second digital data processor in response to an operator request.

82. A method according to claim 79, comprising transferring the first control system component from the first digital data processor to the second digital data processor in response to a request by the second control component.

83. A method according to claim 79, wherein at least the second control component is resident on the second digital data processor.

Appendix

Abstract of the Invention

A control system has blocks or other components that facilitate validation of their own replacements, e.g., downloaded via e-commerce transactions. The system includes first and second process control components. The first component is coupled to a third process control component, with which it transfers information, e.g., as part of an active or ongoing control process. The second component can be, for example, an update or other potential replacement for the first component. The first and/or second components can effect substitution of the second component for the first. More particularly, they can effect coupling of the second component for information transfer with the third component and decoupling of the first component from such transfer with the third component. Preferably, such coupling and decoupling occur while the process control system remains active.

Exchange.3057134.1

Claims :

1. A control device comprising

a virtual machine environment,

the virtual machine environment executing an object or other software construct (collectively, "object") that configures the device to provide a control function,

the object communicating a datum with an entity that stores the datum,

the object accessing the datum by reference to storage maintained in the entity.

2. The control device of claim 1, wherein

the virtual machine environment is a Java virtual machine, and

the object is a Java object.

3. The control device of claim 1, wherein the object accesses the datum via any of a pointer, an address, or a symbolic or other reference to storage maintained in the entity.

4. The control device of claim 3, wherein the object accesses the datum for purposes of any of getting or setting a value thereof.

5. The control device of claim 3, wherein the object does not maintain a copy of the datum.

6. The control device of any of claims 1 - 5, wherein the datum is any of a measurement, a setpoint or other value, or any of a range, a status, a limit status, a time stamp or other information pertaining to such measurement, setpoint or other value.

7. The control device of any of claims 1 - 5, wherein the object configures the control device to provide any of sensing, actuation and other control functions.

8. The control device of any of claims 1 - 5, wherein the object configures the control device to execute one or more control blocks.

9. The control device of claim 8, wherein one or more of the control blocks execute a control algorithm, provide input, and/or provide output.

10. The control device of any of claims 1 - 5, wherein the device is any of a workstation, controller, control station, and a field device.

11. The control device of claims 1 - 5 adapted for providing any of process, industrial, environmental or other control.

12. The control device of claims 1 - 5, wherein the object comprises

one or more mandatory portions for which memory is allocated in the control device at the time of object creation,

and one or more optional parts for which memory space is allocated subsequent to object creation as needed.

13. The control device of claims 1 - 5, comprising

a second object that executes in the virtual machine environment,

the first and second objects having input and output parts defined from subsets of a common set of input and output classes, respectively.

14. The control device of claims 1 - 5, wherein the object configures the device to provide an analog input function with multiple inputs coupled to accept readings from multiple respective input devices and to generate an output based on one or more of those readings.

15. A control system comprising

one or more control devices, each providing a virtual machine environment,

a plurality of objects executing in the one or more virtual machine environments,

the plurality of objects including a first object and a second object that communicate a datum with one another,

the first object maintaining a sole instance of the datum as between at least the first and second objects,

the second object accessing the datum by reference.

16. The control system of claim 15, wherein

the virtual machine environment maintained by the control devices is a Java virtual machine, and

the first object and the second objects are Java objects.

17. The control system of claim 15, wherein the second object accesses the datum via any of a pointer, an address, or a symbolic or other reference to the instance of the datum maintained by the first object.

18. The control system of claim 17, wherein the second object accesses the datum for purposes of any of getting or setting a value thereof.

19. The control system of any of claims 15 - 18, wherein the datum is any of a measurement, a setpoint or other value, or any of a range, a status, a limit status, a time stamp or other information pertaining to such measurement, setpoint or other value.

20. The control system of any of claims 15 - 18, wherein the first and second objects configure control devices on which they execute to provide any of sensing, actuation and other control functions.

21. The control system of any of claims 15 - 18, wherein the first and second objects configure the one or more control devices on which they reside to execute control blocks.

22. The control system of claim 21, wherein at least one of the control blocks executes a control algorithm, provides input and/or provides output.

23. The control system of any of claims 15 - 18, wherein one or more of the control devices is any of a workstation, controller, control station, and a field device.

24. The control system of claims 15 - 18 adapted for providing any of process, environmental or other control.

25. The control system of claims 15 - 18, wherein at least one of the first and second object comprise

one or more mandatory portions for which memory is allocated in the control device at the time of object creation,

and one or more optional parts for which memory space is allocated subsequent to object creation as needed.

26. The control system of claims 15 - 18, wherein the first and second objects have input and output parts defined from subsets of a common set of input and output classes, respectively.

27. The control system of claims 15 - 18, wherein at least one of the first and second objects configures the device to provide an analog input function with multiple inputs coupled to accept readings from multiple respective input devices and to generate an output based on one or more of those readings.

28. A control system comprising

one or more control devices, each providing a virtual machine environment,

a plurality of objects executing in the one or more virtual machine environments,

the plurality of objects including a first object and a second object between which a connection is established to communicate with one another one or more data contained in a data structure,

the first object maintaining a sole instance of the data structure and data contained therein as between at least the first and second objects,

the second object accessing data in the data structure by reference.

29. The control system of claim 28, wherein the data structure comprises a unidirectional variable having

a measurement, a setpoint or other value, and

any of a range, a status, a limit status, a time stamp or other information pertaining to such measurement, setpoint or other value.

30. The control system of claim 29, wherein

the virtual machine environment maintained by the control devices is a Java virtual machine, and

the first object and the second objects are Java objects.

31. The control system of claim 29, wherein the second object accesses data in the data structure via any of a pointer, an address, or a symbolic or other reference to the instance of the data structure maintained by the first object.

32. The control system of claim 29, wherein the second object accesses the data structure for purposes of any of getting or setting a datum thereof.

33. The control system of claim 28, wherein the datum is any of a measurement, a setpoint or other value, or any of a range, a status, a limit status, a time stamp or other information pertaining to such measurement, setpoint or other value.

34. The control system of claim 28, wherein the first and second objects configure control devices on which they execute to provide any of sensing, actuation and other control functions.

35. The control system of claim 28, wherein the first and second objects configure the one or more control devices on which they reside to execute control blocks.

36. The control system of claim 33, wherein at least one of the control blocks executes a control algorithm, provides input and/or provides output.

37. The control system of claim 28, wherein one or more of the control devices is any of a workstation, controller, control station, and a field device.

38. The control system of claim 28 adapted for providing any of process, environmental or other control.

39. The control system of any of claims 28 - 38, wherein the data structure comprises a bi-directional variable having

a forward-going measurement, setpoint or other value (collectively, "forward-going value"),

any of a range, a status, a limit status, a time stamp or other information pertaining to the forward-going value,

a back-going measurement, setpoint or other value (collectively, "back-going value"), and

any of a range, a status, a limit status, a time stamp or other information pertaining to the back-going value).

40. A control system comprising

one or more control devices, each providing a virtual machine environment,

a plurality of objects executing in the one or more virtual machine environments,

the plurality of objects including a first object that communicates a datum with a second object and a third object,

the first object maintaining a sole instance of the datum as between at least the first, second and third objects,

the second and third objects accessing the datum by reference.

41. A control system comprising

one or more control devices, each providing a virtual machine environment,

a plurality of objects executing in the one or more virtual machine environments,

the plurality of objects including a first object with which connections are established by a second object and by a third object, each connection being for the transfer of data contained in a data structure,

the first object maintaining a sole instance of the data structure and data contained therein as between at least the first, second and third objects,

the second and third objects accessing data in the data structure by reference.

42. The control system of any of claims 40 - 41, wherein the data structure comprises a bi-directional variable having

a forward-going measurement, setpoint or other value (collectively, "forward-going value"),

any of a range, a status, a limit status, a time stamp or other information pertaining to the forward-going value,

a back-going measurement, setpoint or other value (collectively, "back-going value"), and

any of a range, a status, a limit status, a time stamp or other information pertaining to the back-going value).

43. A method of operating a control device comprising

executing an object or other software construct (collectively, "object") on a virtual machine environment provided in the control device, the object configuring the control device to provide a control function,

communicating a datum between the object and an entity that stores the datum, the object accessing the datum by reference to storage maintained in the entity.

44. The method of claim 43, wherein the virtual machine environment is a Java virtual machine, and wherein the object is a Java object.

45. The method of claim 43, comprising accessing the datum from the object via any of a pointer, an address, or a symbolic or other reference to storage maintained in the entity.

46. The method of claim 45, comprising accessing the datum for purposes of any of getting or setting a value thereof.

47. The method of claim 45, wherein the object does not maintain a copy of the datum.

48. The method of any of claims 43 - 47, wherein the datum is any of a measurement, a setpoint or other value, or any of a range, a status, a limit status, a time stamp or other information pertaining to such measurement, setpoint or other value.

49. The method of any of claims 43 - 47, comprising executing the object to configure the control device to provide any of sensing, actuation and other control functions.

50. The method of any of claims 43 - 47, wherein the executing step includes executing one or more control blocks associated with the object.

51. The method of claim 50, wherein one or more of the control blocks comprises a control algorithm, an input function, and/or an output function.

52. The method of any of claims 43 - 47, wherein the device is any of a workstation, controller, control station, and a field device.

53. The method of claims 43 - 47 adapted for providing any of process, industrial, environmental or other control.

54. A method of operating a control system comprising

executing a plurality of objects in virtual machine environments provided on one or more control devices, the objects configuring the one or more devices to provide control functions,

communicating a datum between first and second ones of the plurality of objects,

maintaining in the first object a sole instance of the datum, as between at least the first and second objects, and

accessing by reference the datum with the second object.

55. The method of claim 54, wherein the virtual machine environment is a Java virtual machine, and the plurality of objects are Java objects.

56. The method of claim 54, wherein the accessing step includes accessing the datum via any of a pointer, an address, or a symbolic or other reference to the instance of the datum maintained by the first object.

57. The method of any of claims 54 - 56, wherein the datum is any of a measurement, a setpoint or other value, or any of a range, a status, a limit status, a time stamp or other information pertaining to such measurement, setpoint or other value.

58. The method of any of claims 54 - 56, wherein the executing step includes executing at least the first and second objects to configure the control devices to provide any of sensing, actuation and other control functions.

59. The method of any of claims 54 - 56, wherein the executing step includes executing one or more control blocks associated with the first and second objects.

60. The method of claim 59, wherein at least one of the control blocks executes a control algorithm, provides input and/or provides output.

61. The method of any of claims 54 - 56, wherein one or more of the control devices is any of a workstation, controller, control station, and a field device.

62. The method of claims 54 - 56 adapted for providing any of process, environmental or other control.

63. A method of operating a control system, comprising

executing a plurality of objects in virtual machine environments provided on one or more control devices, the objects configuring the one or more devices to provide control functions,

communicating one or more data contained in a data structure between first and second ones of the plurality of objects,

maintaining in the first object a sole instance of the data structure and data contained therein as between at least the first and second objects,

accessing by reference data in the data structure with the second object.

64. The method of claim 63, wherein the data structure comprises a unidirectional variable having

a measurement, a setpoint or other value, and

any of a range, a status, a limit status, a time stamp or other information pertaining to such measurement, setpoint or other value.

65. The method of claim 63, wherein the executing step includes executing at least the first and second objects to configure the control devices to provide any of sensing, actuation and other control functions.

66. The method of claim 63, wherein the executing step includes executing one or more control blocks associated with the first and second objects.

67. The method of claim 63, wherein at least one of the control blocks executes a control algorithm, provides input and/or provides output.

68. The method of claim 63, wherein one or more of the control devices is any of a workstation, controller, control station, and a field device.

69. The method of claim 63 adapted for providing any of process, environmental or other control.

70. The method of any of claims 63 - 69, wherein the data structure comprises a bi-directional variable having

a forward-going measurement, setpoint or other value (collectively, "forward-going value"),

any of a range, a status, a limit status, a time stamp or other information pertaining to the forward-going value,

a back-going measurement, setpoint or other value (collectively, "back-going value"), and

any of a range, a status, a limit status, a time stamp or other information pertaining to the back-going value).

71. A method of operating a control system, comprising

executing a plurality of objects in virtual machine environments provided on one or more control devices, the objects configuring the one or more devices to provide control functions,

communicating one or more data contained in a data structure between first one of the objects and second and third ones of the objects,

maintaining in the first object a sole instance of the data structure and data contained therein as between at least the first, second and third objects,

accessing by reference data in the data structure with the second and third objects.

72. The method of claim 71, wherein the data structure comprises a bi-directional variable having

a forward-going measurement, setpoint or other value (collectively, "forward-going value"),

any of a range, a status, a limit status, a time stamp or other information pertaining to the forward-going value,

a back-going measurement, setpoint or other value (collectively, "back-going value"), and

any of a range, a status, a limit status, a time stamp or other information pertaining to the back-going value).

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/20828

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) : G05B 15/00
US CL : 700/1

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 700/1,9,86;709/1

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EAST

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X,P	US 6,002,104 A (HSU) 14 December 1999 (14.12.1999), column 2 lines 20-30, column 3 lines 51-55, column 7 lines 46-58, column 10 lines 65-67, column 12 lines 1-20	1-5,9,15-18,22,28-38,40,41,43-47,51,54-56,60,63-69,71,72
A	US 5,854,750 A (PHILLIPS et al) 29 December 1998 (29.12.1998)	All
A	US 5,844,804 A (SCHUSSLER) 1 December 1998 (01.12.98)	All
A	US 6,085,120 A (SCHWERDTFEGER et al) 4 July 2000 (04.07.2000)	All

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 October 2000 (13.10.2000)	15 NOV 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703)305-3230	Authorized officer Edward (Skip) Gain James R. Matthews Telephone No. 703-305-7335

Form PCT/ISA/210 (second sheet) (July 1998)

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/20828

Box I Observations where certain claims were found unsearchable (Continuation of Item 1 of first sheet)

This international report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claim Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claim Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:

3. ☒ Claim Nos.: 6-8,10-14,19-21,23-27,39,42,48-50,52,53,57-59,61,62 and 70
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of Item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.
☐ No protest accompanied the payment of additional search fees.